UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jernej Plešnar

Prototip rešitve za vodenje brezpilotnega letala z mobilno aplikacijo

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: viš. pred. Igor Rožanc

Ljubljana, 2016

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V diplomski nalogi izdelajte prototip celovite rešitve za vodenje brezpilotnega letala s pomočjo mobilne aplikacije, ki deluje v različnih mobilnih omrežjih. Rešitev naj obsega štiri dele: strojni del na osnovi Raspberry Pi sistema s kamero, Wi-Fi sprejemnikom in 4G modemom, programski del tega sistema z ustrezno medprocesno komunikacijo, komunikacijski del na osnovi P2P WebRTC komunikacije ter mobilno Android aplikacijo, ki preko preprostega uporabniškega vmesnika omogoča upravljanje letala s prenosom videa v realnem času. Rešitev razvijte sistematično od ideje do delujočega izdelka s preverjanjem različnih alternativnih delnih rešitev, pri čemer naj bo poudarek na modularnosti, funkcionalnosti, učinkovitosti in cenovni dostopnosti. Delovanje rešitve preverite v praksi na modelu letečega krila.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Jernej Plešnar sem avtor diplomskega dela z naslovom:

Prototip rešitve za vodenje brezpilotnega letala z mobilno aplikacijo

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. Igorja Rožanca,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek(slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 8.9.2016 Podpis avtorja:

Rad bi se zahvalil vsem, ki so mi kakorkoli pomagali pri diplomski nalogi, še posebej mentorju viš. pred. Igorju Rožancu, ki mi je svetoval, me podpiral in verjel v mojo idejo. Zahvalil bi se tudi svoji družini in letos avgusta pokojnemu očetu, prijateljem in vsem, ki so kakorkoli del mojega življenja in so me posredno ali neposredno podpirali in mi pomagali.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
API	Application Programming Interface	Programski vmesnik aplikacije
COM	Communication port	Komunikacijska vrata
CPU	Central Processing Unit	Centralna procesna enota
CSI	Camera Serial Interface	Serijsko vodilo za kamero
ESC	Electronic Speed Controller	Krmilnik brezkrtačnih elektromotorjev
FPV	First Person View	Prvoosebni pogled
GIL	Global Interpreter Lock	Globalna ključavnica tolmača
GPIO	General-Purpose Input/Output	Splošno namenski vhod/izhod
GPS	Global Positioning System	Sistem globalnega določanja položaja
GPU	Graphics Processing Unit	Grafični procesor
HAT	Hardware Attached on Top	Razvojni dodatek za RPi
HDMI	High-Definition Multimedia Interface	Multimedijski vmesnik visoke ločljivosti
HUD	Head-Up Display	Vrsta transparentnega prikazovalnika
I/O	Input/Output	Vhod/izhod
I2C	Inter-Integrated Circuit	Protokol komunikacije med vezji
ICE	Interactive Connectivity Establishment	Protokol za interaktivno vzpostavljanje povezave
IDE	Integrated Development Environment	Integrirano razvojno okolje
IoT	Internet of Things	Internet stvari
IP	Internet Protocol	Omrežni internetni protokol

IPC	Inter-Process Communication	Medprocesna komunikacija
NAT traversal	Network Address Translation traversal	Preslikava omrežnih naslovov
OS	Operating System	Operacijski sistem
P2P	Peer-to-Peer	Direktna omrežna komunikacija med dvema napravama
PWM	Pulse-Width Modulation	Pulzno-širinska modulacija
R&D	Research and Development	Raziskava in razvoj
RAM	Random Access Memory	Bralno-pisalni pomnilnik
RPi	RPi microcomputer	Mikroračunalnik Raspberry Pi
RTC	Real Time Communication	Komunikacija v realnem času
SDK	Software Development Kit	Paket za razvoj programske opreme
SoC	System on Chip	Integrirano vezje, ki računalniški sistem združuje v enem čipu
SSH	Secure Shell	Varna lupina
SSL/TLS	Secure Sockets Layer/ Transport Layer Security	Plast varnih vtičnikov
STUN	Simple Traversal of UDP through NATs	Enostavno usmerjanje skozi NAT po protokolu UDP
TURN	Traversal Using Relays around NAT	Usmerjanje s pomočjo posrednikov mimo NAT
UDP	The User Datagram Protocol	Nepovezovalni protokol
USB	Universal Serial Bus	Univerzalno serijsko vodilo
VNC	Virtual Network Computing	Sistem za oddaljen dostop do grafičnega sistema
WebRTC	Web Real-Time Communication	Komunikacija v realnem času za splet
Wi-Fi	Wireless local area network	Brezžično lokalno omrežje
X	X Window System	Okenski sistem X

Povzetek

Naslov: Prototip rešitve za vodenje brezpilotnega letala z mobilno aplikacijo

V diplomski nalogi je predstavljen razvoj rešitve za oddaljeno vodenje brezpilotnega letala s pomočjo prvoosebnega pogleda, ki za komunikacijo uporablja mobilno omrežje. Ta rešuje problem omejenosti z dosegom, cenovne dosegljivosti in enostavnosti za uporabo. Za doseg tega cilja je bilo potrebno mnogo korakov pri analizi zahtev, načrtovanju posameznih delov rešitve in celotnega sistema, raziskovanju ustreznih orodij, učenju programskih jezikov in programskih orodij, implementaciji in testiranju posameznih delov. Rešitev je sestavljena iz komunikacijskega dela, programskega in strojnega dela. Prvi obsega rešitev s pomočjo WebRTC tehnologije in strežnikov z Linux sistemom, drugi pa aplikacijo za sistem Android in Python program za mikrokrmilnik Raspberry Pi. Tretji pa je izveden z mikrokrmilnikom Raspberry Pi z dodatki (razširitvena vezja, kamera, GPS, modem) in ohišjem. Rešitev je bila testirana na letečem krilu v varnem okolju.

Ključne besede: oddaljeno vodenje brezpilotnega letala, krmilni sistemi, Raspberry Pi, medprocesna komunikacija, WebRTC, prenos videa v realnem času, Android

Abstract

Title: Mobile application controlled unmanned aerial vehicle solution prototype

The following thesis describes unmanned aerial vehicle control solution developement based on mobile connection. It solves range limitation, price availability and use complexity. Major steps include requirement analysis, software modules and system planning, tools and developement process research, software implementation, hardware integration, programming languages, tools and developement kits learning, implementation and modular testing of a system. The solution combines communication, software and hardware part. The first part is based on WebRTC technology and Linux servers`s solution. The second part is performed by Android based application with a Raspberry Pi running Python software, whiile the third part contains a Raspberry Pi based hardware and associate components in a wooden case. Everything is tested on a flying wing in a safe environment.

Keywords: remote controlled unmanned aerial vehicle, control systems, Raspberry Pi, interprocess communication, WebRTC, video streaming in real time, Android

Poglavje 1 Uvod

V diplomski nalogi bomo predstavili razvoj prototipa od motivacije in idejne zasnove do raziskovanja, razvoja in testiranja rešitve. Pri tem bomo kratko in jedrnato opisali ključne korake, podrobneje pa opisali dele, kjer je bilo največ odkritega, razvitega in ugotovljenega.

Hoteli smo narediti sistem za brezpilotno letalo tako, da ga bo možno voditi s pomočjo mobilne aplikacije, samo letalo pa bo upravljal mikrokrmilnik Raspberry Pi. Pri tem mora biti sistem sposoben prenašati video tok, podatke o senzorjih in ukaze dovolj hitro, da je rešitev zanesljiva in uporabna.

Motivacija izvira iz zanimanja za tovrstna področja, ki se prepletajo pri razvoju letečih robotov. Predvsem si želimo spoznati uporabnosti in zmogljivosti sistema Raspberry Pi, si postaviti izziv, ki ga ni mogoče enostavno rešiti, težavo razrešiti z inženirskim pristopom in delitvijo na manjše kose. S pomočjo analize, ocenjevanja, načrtovanja, raziskovanja, razvoja in testiranja želimo uspešno priti do ustreznih zaključkov tako pri posameznih sklopih, kot pri celotni rešitvi.

Zanima nas več področij znanosti in tehnologije, še zlasti letalstvo, aeronavtika in aerodinamika. Vsa tri področja poznamo na zelo amaterskem nivoju. Več znanj imamo na področju računalništva (angl. computer science). Pred nekaj leti smo naredili prvi korak v povezovanju teh stvari, ko smo se samostojno ukvarjali s projektom avtonomnega letala z mikrokrmilnikom Arduino. Poskus se je sicer končal neuspešno s stališča samega projekta, saj ni bil niti konceptualno niti finančno dobro podprt. Neuspeli poskus pa je prinesel veliko znanja, ker smo ogromno stvari delali od začetka, kar pa z inženirskega, predvsem pa podjetnega vidika nikakor ni dobra praksa. Naučili smo se predvsem to, da si je včasih pri delu potrebno priznati, da projektu ne gre najboljše, ker ni bil dobro zasnovan že v načrtu, bodisi ker ima pomanjkljivosti, ker izumljamo že znane stvari, ali kaj tretjega, in si priznati, da projekt lahko tudi ne uspe.

Nekaj let zatem smo poleg rednega dela na fakulteti obiskovali trimesečni spletni tečaj aeronavtike na odprtokodni platformi za spletno učenje Edx, saj nas je zanimalo področje aerodinamike in ga uspešno opravili ter pridobili certifikat [1]. Naše zanimanje je vodilo k enem izmed interdisciplinarnih projektov na temo brezpilotnih letal kmalu zatem pa

soudeležbo na mednarodnem tekmovanju DBF [2], kjer smo aprila 2015 z ekipo 20 članov iz Ljubljanske univerze osvojili zmago v Tucsonu, ZDA.

Z zanimanjem za teorijo delovanja raket, letal, satelitov, roverjev, ostalih robotov in projektov organizacij NASA (angl. National Aeronautics and Space Administration) in ESA (European Space Agency), predvsem pa avtonomnih brezpilotnih letal je nastopilo intenzivnejše obdobje zanimanja za mikrokrmilnike. Tako smo se hoteli poglobiti v novi, popularni in zmogljivi mikrokrmilniški sistem Raspberry Pi, ki je sposoben poganjati operacijski sistem Linux. Potrebno je bilo osvojiti, obnoviti in poglobiti znanje ukazov, samega sistema Linux, razvoja za operacijski sistem Android, razvoja v orodju Xamarin za Android, programskih jezikov C#, Javascript in Python, arhitekturnih in razvijalskih vzorcev, minimalnih elektrotehniških osnov, komunikacijskih omrežij, organizacije in vzdrževanja kode in ogromno ostalih inženirskih, miselnih, logičnih in programerskih izzivov in veliko razvojno-raziskovalnega dela na poti do rezultata.

Omejitve so nam predstavljali izzivi, časovna zahtevnost reševanja težav, zahteve arhitekture (komponent) in povezave med posameznimi deli, finančne omejitve, zakonske omejitve in pravila o brezpilotnih letalih, varnostne meje in zahteve, odgovornost za zanesljivost delovanja rešitve in njegove varne uporabe za nekomercialne namene na ruralnem območju in pomanjkanje znanja in virov na tem področju. Na začetku se je poznalo predvsem pomanjkanje komponent in težavnost razvoja brez ustreznih pripomočkov in delujočih podsistemov.

Razvoj rešitve bomo predstavili sistematično. Najprej si bomo pogledali, kako smo analizirali problem, ki ga moramo rešiti, tehnike in metode, za katere smo se pri tem odločili, obstoječe rešitve in njihove pomanjkljivosti in tehnologijo, ki nam razvoj rešitve omogoča. Potem bomo podrobneje opisali potek načrtovanja, raziskovanja in razvoja posamezne faze ali sklopa sistema, ki predstavlja rešitev. S kratkim poglavjem bomo preverili prve korake idejne zasnove, nato pa se poglobili v sklop strojne opreme rešitve, programske rešitve na RPi, programske rešitve na mobilni napravi in programske rešitve komunikacijskega dela. Na koncu si bomo pogledali še potek razvoja modela letala od modeliranja računalniškega CAD modela pa do fizičnega prototipa, uporabljene tehnologije pri samem razvoju in kaj smo se iz vsega tega naučili in na kratko opisali naša mnenja. Sledol bo kratek pregled porabe časa, možne nadgradnje in izboljšave rešitve in njenih pomanjkljivosti.

Poglavje 2 Analiza problema

Za razumevanje ciljev rešitve si bomo najprej pogledali podobnosti obstoječih rešitev, predvsem njihove prednosti in slabosti glede na našo nalogo. Nato se bomo poglobili v samo nalogo, predstavili namen, smernice in tehnologijo, ki jo bomo potrebovali.

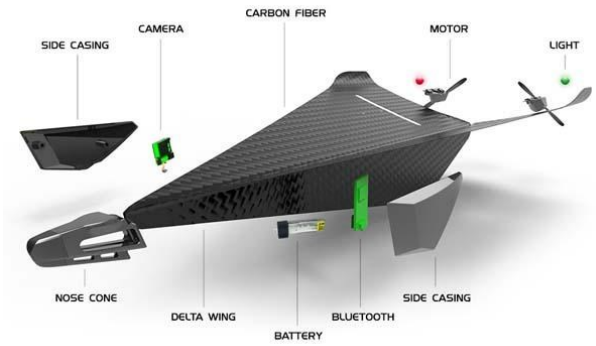
2.1 Raziskava obstoječih rešitev

V tabeli 1 bomo pogledali obstoječe komercialne rešitve in motivacijske projekte drugih.

Rešitev	Opis	Prednosti	Slabosti
Carbon flyer [3]	Mobilno vodeno letalo, prikaz na sliki 1	Karbonska vlakna, lahek, odporen proti udarcem, prvoosebni pogled	Kratek doseg, okrnjeno letenje – zavijanje z motorji
Power Up FPV [4]	Mobilno voden dodatek za papirnata letala	Prvoosebni pogled, fotografiranje, lahek, video in avdio snemanje	Kratek doseg (Wi-Fi), okrnjeno letenje – zavijanje z motorji
Pixhawk [5]	Linux odprtokodni avtopilot	Učinkovit operacijski sistem, razširljivost, odprtokodnost, odličen za razvijalce, hobi projekte	»Samo« avtopilot, ne preveč zahteven, a tudi ne čisto enostaven za nepoznavalce
4Gmetry III [6]	Dodatek za Pixhawk avtopilota - heterogenega superračunalnika ODROID-XU4 [30]	Doseg omrežja 4G, prvoosebni pogled, OpenVPN, lahek, ROS (angl. robot OS), odprtokodnost, razširljivost, za razvijalce računalniškega vida	Dodatna enota poleg obstoječih komponent, za nepoznavalce tehnološkega področja neustrezen
C-Astral [7]	Visoko-tehnološki celotni brezpilotni	Vzdržljivost, zmogljivost, tehnologija, doseg, oprema	Cena – nekaj 10 tisoč evrov, ni za povprečno uporabo

	sistemi		
Emlid [8]	Avtopilot na vezju, dodatek za RPi	Podpira Pixhawk, prvoosebni pogled, razširljivost,	Ni enostaven, potrebno poznavanje osnov OS Linux in računalništva
Erle-Brain 2 [9]	Avtopilot na osnovi računalnika Odroid [30] in ROS (angl. robot OS)	Odličen za razvijalce, razširljivost, integrirana kamera	Ni enostaven, potrebno poznavanje osnov OS Linux in računalništva
Mirco drone 3.0 [10]	Zelo uspešna 3,5 miljonski projekt. Majhen pameten quadcopter	FPV, mobilno in radijsko voden, enostaven, lahek - 56g	Kratek doseg, ni razširljivosti, eno-namenski
Parrot AR drone 2.0 [11]	Mobilno voden UAV	Enostaven, lahek, cenovno dostopen, prvoosebni pogled	Kratek doseg, ni razširljivosti, eno-namenski
Skydrone FPV [12]	FPV 4G sistem	Nizka latenca, polna kvaliteta video (1080p)	Samo kamera

Tabela 1: Pregled obstoječih rešitev na področju brezpilotnih letal



Slika 1: Carbon flyer

Nekaj motivacije je bilo tudi iz uspešnih projektov, ki so izvedeni na bolj ali manj inženirske načine.

- mobilna aplikacija za deljenje mobilnega omrežja [13];
- FPV prenos slike s pomočjo RPi [14];
- telemetrija s pomočjo usmerjevalnika, avtopilota in RPi na letalu [15];
- spletni članek z idejo [16].

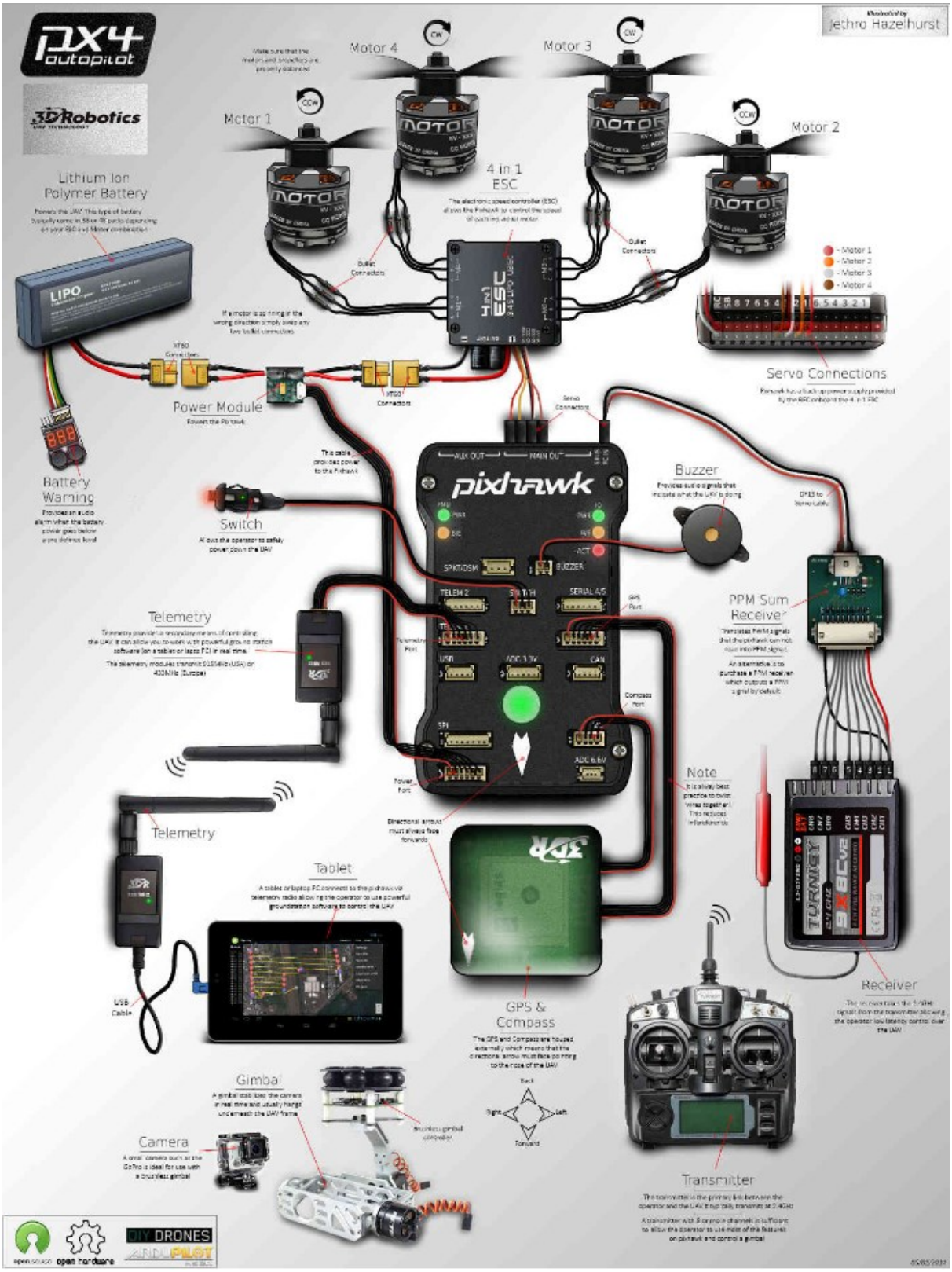
Za motivacijo so poskrbeli tudi uspešni projekti množičnega financiranja (angl. crowdfunding) na osnovi RPi, ki nimajo opravka z UAV. Dva od teh vključujeta multimedijski predvajalnik Slice [17] in komplet za snemanje živali v naravnem okolju [18], ki dokažeta, da je RPi zrel tudi za komercialne rešitve in ne samo za učenje programiranja kot je bil sprva zasnovan in definiran s strani njegovih ustvarjalcev.

Obstoječe rešitve brezpilotnih naprav za podatkovno in video komunikacijo, nadzor in telemetrijo uporabljajo analogne-radijske rešitve na frekvenčnem območju 433 Mhz (EU) ali 915 MHz (ZDA) za telemetrijo, 2.4GHz za krmiljenje in 5.8 GHz za video, (ali 2.4 GHz povezavo za vse skupaj s pomočjo standardnega Wi-Fi protokola). Večja frekvenca zagotavlja večjo pasovno širino in s tem večjo zanesljivost, manjša frekvenca pa zagotavlja večji doseg pri istih pogojih. Radijski protokoli v osnovi ne zagotavljajo zanesljivosti po principu potrjevanja, kot to deluje pri omrežnih protokolih (TCP) pri TCP/IP modelu [147].

Za delovanje potrebujemo tako sprejemnik kot oddajnik, ti pa se lahko med seboj motijo - zato se uporabljajo različne frekvence. Dodatni sprejemniki (predvsem pa oddajniki) predstavljajo dodatno težo, prostor, načeloma neugoden vir toplote, višjo ceno in vzdrževanje, dodatne baterije itd. Za popolnoma avtonomni sistem brezpilotnega letala potrebujemo:

- krmilni sprejemnik in oddajnik,
- kamero z video sprejemnikom in oddajnikom,
- avtopilotni krmilni sistem z GPS sprejemnikom,
- 2 ali celo 3 ločene vire napajanja.

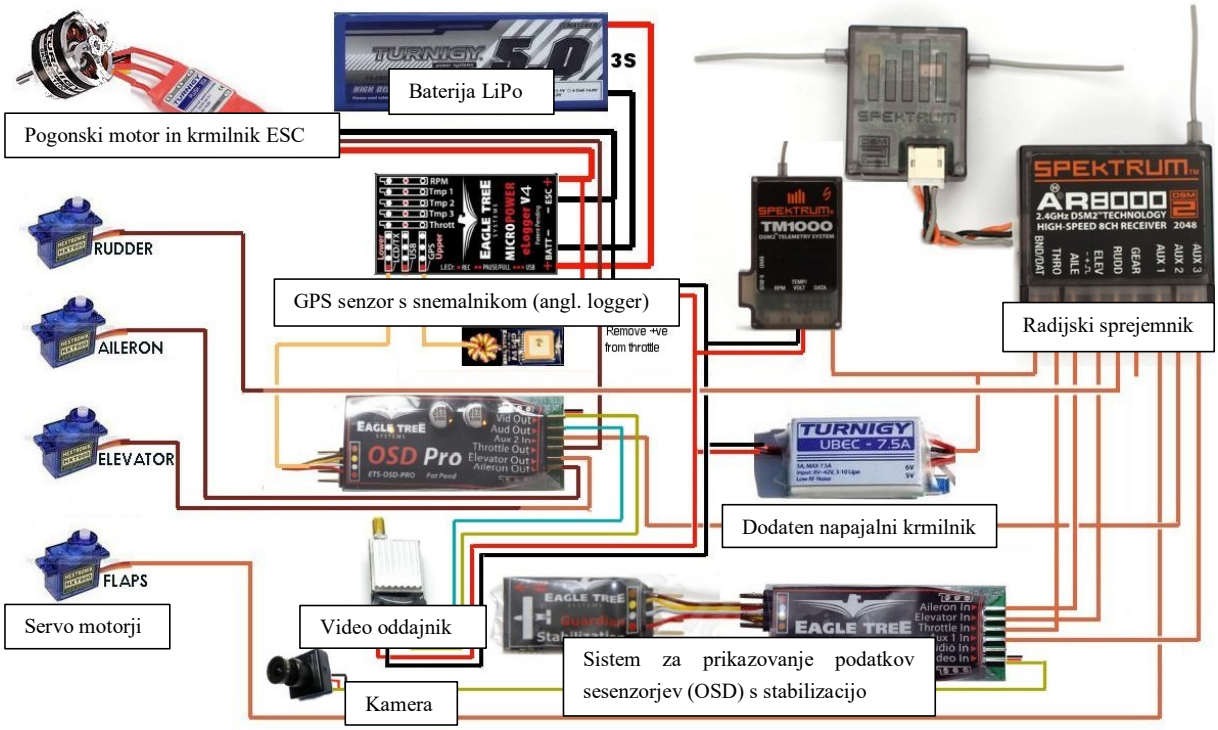
Za daljši doseg potrebujemo tudi usmerjeno anteno, ki jo lahko vodimo sami ročno ali za to namenjen samodejni sledilnik (ki primerja GPS lokaciji obeh sistemov).



Slika 2: Prikaz razširljivosti in povezav sistema Pixhawk [145]

Slika 2 prikazuje zmogljiv sistem brez sistema FPV. Vidimo številne različne komponente, ki jih moramo izbrati, kupiti, jih povezati. Shemo povezav notacije prikazuje slika 3.

Potrebno je tudi nekaj osnovnega znanja elektrotehnike ali vsaj strokovnih izrazov, kar pa je za večino uporabnikov prezahtevno in tako finančno, kot tudi časovno predrago. S tako opremo hitro pridemo do par tisoč evrov, brez avtopilota pa je tudi nujno znanje pilotiranja brezpilotnega zračnega plovila. Nekatere komercialne rešitve omogočajo enostavno uporabo izdelkov, ki pa so za kakršnokoli resno uporabo nepraktični in nezmogljivi, nemodularni ali neprilagodljivi. Druge rešitve so zelo zmogljive in omogočajo uporabo z raznimi senzorji, vendar je potrebnega poznavanja letalstva, elektrotehnike in računalništva enostavno preveč.



Slika 3: Tipična oprema in povezave med njimi [146]

2.2 Namen

Z našo rešitvijo poskušamo prinesti prednosti brezpilotnih letal s pomočjo sodobne tehnologije slehernemu tehnološkemu ali letalskemu navdušencu. Ključne prednosti so:

- neomejen doseg (glede na pokritost s signalom, to smo preverili za slovenska omrežja Simobil [19] in Mobitel [20]);
- razširljivost na vse vrste brezpilotnih letal in modularnost;

- edinstvenost in enostavnost FPV izkušnje na lastni mobilni napravi s pomočjo intuitivnih kontrol z nagibi in nazornimi ukazi.

Hrati pa orodja avtopilota omogočajo enostavno in sproščeno letenje. Ni potrebna izobrazba elektrotehnike ali računalništva in programiranja niti zahtevno branje navodil in dokumentacije.

2.3 Pomanjkljivosti

Pomanjkljivost oziroma slabost rešitve je zaenkrat še predvsem nepreizkušenost. Tu je še plačljiv prenos podatkov zaradi uporabe storitev mobilnega operaterja, cena pa je odvisna od naročniškega paketa. Še vedno pa je rešitev tako s stališča enostavnosti, pa tudi cenovno, vsekakor v prednosti. Količina zakupljenega mobilnega prenosa podatkov se povečuje, cene nižajo, cene gostovanja po EU so se oziroma se bodo kmalu poenotile. Prav tako naj bi leta 2021 prišel mobilni standard 5G [21] in s tem večja pokritost, hitrosti in nižje latence [22].

2.4 Smernice

Rešitev želimo izdelati tako, da bo njena uporabnost splošna. Le malo spremenjen program lahko z nekaj znanja programiranja omogoči uporabo dveh pogonskih motorjev, dodan modul pa lahko omogoči uporabo drugačne konfiguracije letala, z drugačnim številom krmilnih površin, zakrilc, zračnimi zavorami, aktuatorji, zložljivimi kolesi, mehanizmi za usmerjenje kamere in drugih pripomočkov (dodatnih anten, senzorjev, sončnih celic). Lahko se uporabijo različni pogoni - tako več elektro pogonov, kot tudi možnost motorjev z notranjem izgorevanjem in reaktivnih in raketnih vrst motorjev. Z nekaj truda bi bilo možno sprogramirati večuporabniško rešitev nad istim izdelkom, vendar se zaenkrat razvoj glede na potrebo in zahtevnost ne zdi relevanten. Možne razširitve so tudi za druge vrste vozil - avtomobilov, podmornic, ladij, helikopterjev, večrotornikov in tudi bolj ekstremnih vozil, kot je x-plane, hoovercraft in raketa.

2.5 Potrebna tehnologija

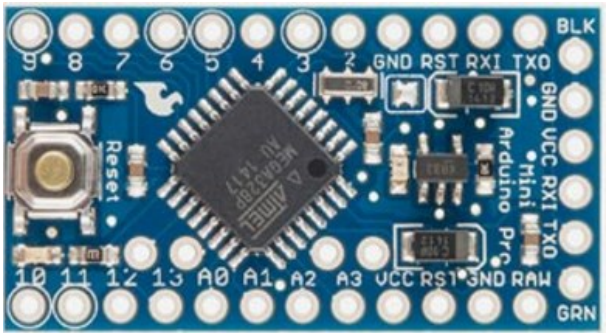
V naslednjih treh delih bomo opisali, kaj nam razvoj take rešitve omogoča s tehnološkega vidika. V ta namen bomo najprej opisali krmilne sisteme, da razumemo njihov princip delovanja, saj so eden glavnih delov, ki jih bomo potrebovali. Nato si bomo na kratko pogledali statistiko uporabnikov mobilnih aplikacij in katera infrastruktura omogoča komuniciranje obeh.

2.5.1 Krmilni sistemi

2.5.1.1 Splošno

Mikrokrmilniki so naprave, brez katerih življenje v današnjem svetu ni mogoče. Hitro, neprestano in učinkovito lahko upravljajo pralne stroje, avtomobile, pametne hiše, peči za centralno kurjavo, letala, naprave v vesolju, satelite in tudi mnoge naprave, ki so nam samoumevne, da so nam na voljo, kot so kavni avtomati in vremenske postaje [148].

Obstaja nekaj nam znanih proizvajalcev mikrokrmilnikov in mikrokrmilniških razvojnih plošč: Arduino [23], Texas Instruments [24], Microchip PIC [25], STM, [26] RaspberryPi [27], Intel [28], Beaglebone [29], Odroid [30].



Slika 4: Arduino Pro Mini [23]


Slika 5: Razvojna plošča Odroid-C2 [30]

Slika 4 prikazuje popularni izdelek Arduino Pro Mini, slika 5 pa manj znan izdelek Odroid-C2, ki je po zmogljivostih boljši od Raspberry Pi 2.

Mikrokrmilnik sprejema podatke iz vhodnih naprav in krmili izhodne naprave glede na specifičen program, ki ga izvaja. Vhode predstavljajo senzorji, ki na podlagi lastnosti določenih materialov prevajajo določeno količino električno energije. Tako na podlagi padca napetosti dobimo na vhod vrednosti, ki po pretvorbi iz analogne v digitalno obliko predstavljajo količino svetlobe na senzorju, temperaturo senzorja, tlak na senzorju, stanje senzorja (gumb), vlažnost zraka v senzorju. Izhode predstavljajo razni aktuatorji, ki opravljajo neko nalogo. To so lahko razni elektromotorji, tranzistorji, LED (svetleče) diode itd.

2.5.1.2 Krmilni sistemi brezpilotnih letal

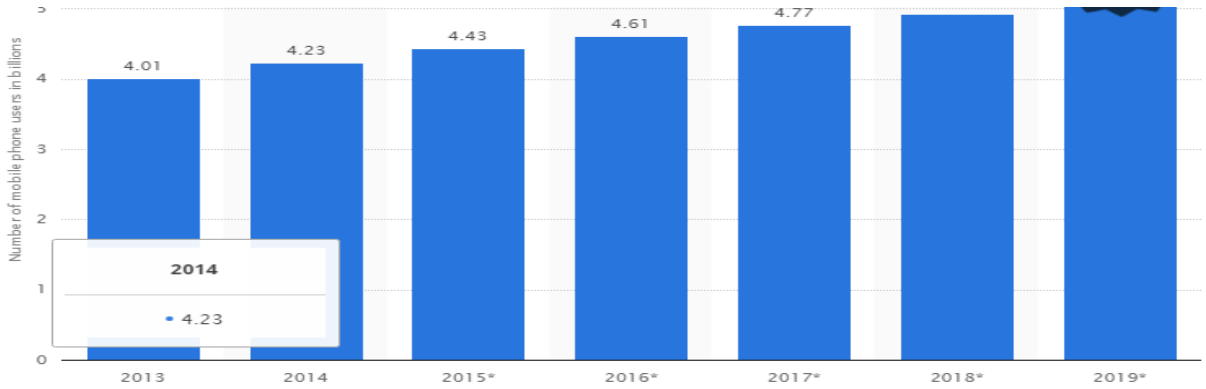
Za namen prikaza obstoječih robustnih rešitev bomo navedli le sisteme, ki so sposobni za avtonomnost in širok nabor funkcionalnosti. Izpustili bomo manj zmogljive krmilnike, ki se prav tako uporabljajo zgolj za namene stabilizacije v izvedbi samega vezja brez programske opreme.

Za potrebe amaterskega in profesionalnega modelarstva se najbolj uporablja Pixhawk [5]. Ostali znani pa so: FY Panda II [31], DJI Naza [32], Piccolo [33] in Hse UAV autopilot [34].

Brezpilotna letala uporabljajo podobne senzorje kot pravi letalski sistemi, vendar namenu in cenovni dostopnosti primerne velikosti, kakovosti in natančnosti. Prav tako se uporabljajo temperaturni in vlažnostni senzorji, barometri za merjenje tlaka in izračun višine, pitotova cev za merjenje zračne hitrosti, žiroskopi in merilniki pospeška za merjenje pospeškov in nagibov letala, GPS sistemi za določanje lokacije letala, merilniki električnega toka itd.

2.5.2 Mobilne aplikacije in delež uporabnikov mobilnih naprav

Brez uporabnikov mobilne tehnologije in dodelanih razvojnih orodij ter znanj, ki nam omogočajo razvoj programske opreme, bi bila rešitev težko izvedljiva in zelo draga. V današnjem času imamo vsaj eno mobilno napravo, s katero smo povezani na internetno omrežje. Formalno in neformalno znanje je vedno bolj dostopno, prav tako pa se povečuje dostopnost tehnoloških komponent, ki so bile včasih nedosegljive. Leta 2015 je mobilne naprave uporabljalo 4.77 milijard ljudi, kar lahko vidimo na sliki 6.



Slika 6: Število uporabnikov mobilnih naprav [149]

2.5.3 Infrastruktura in tehnologija v oblaku

Oblačne storitve (kot jih ponujata podjetji Amazon in Microsoft) nam omogočajo enostavno, učinkovito in varno uporabo infrastrukture in storitev na strežnikih, ki zanesljivo delujejo 99,95% časa [35] in nam olajšajo postavitev virtualnega strežnika s poljubnim operacijskim sistemom.

Virtualni strežniki in servisi nam lahko služijo kot posredniki pri komunikaciji v različnih omrežjih, medtem pa lahko ogromno količino podatkov tudi shranjujejo in na več načinov obdelujejo. Nam bojo strežniki omogočali izvedbo komunikacijske rešitve.

Poglavje 3 Načrtovanje, raziskovanje in razvoj

3.1 Idejne zasnove in prve ideje

Ker idejna zasnova ne temelji na P2P komunikaciji, je temu primerna začetna arhitektura programske opreme. Mogoče ni običajen način v končnem izdelku opisovati opuščene ali spremenjene načrte, ampak tako želimo pokazati razliko pogleda na problem v začetnih fazah projekta, med projektom in na koncu projekta. S tem bi radi pokazali, kako je razvoj dinamičen in se spreminja, hkrati pa se pokaže tako časovno kot tudi razvojno zahtevnost izdelave take rešitve. Skozi diplomsko delo bomo lahko tako ocenili tudi količino naučenega znanja in pridobljenih izkušenj ter cenili vrednost tako same rešitve, kot tudi diplomskega dela.

Pri branju idejne zasnove bodimo pozorni, da so to samo prvotni, lahko bi rekli primitivni načrti, ki ne bi nikakor delovali na ustrezen način. Sledijo uporabniške zgodbe, ki definirajo cilje oziroma zahteve končnega uporabnika, kaj je pričakovano oziroma mora biti implementirano. Za tem bomo bolj tehnično preverili funkcionalnosti aplikacije, strojne opreme rešitve, njeno sestavo in sestavo letala kot celote.

3.1.1 Rešitev in njene komponente

Vsak del rešitve na nek način predstavlja fizično ločen del izdelka, ki je bil ločeno izveden. Na eni strani je strojna oprema z RPi, ki krmili strojno opremo, na drugi pa mobilna aplikacija na mobilni napravi s sistemom Android, ki omogoča njeno krmiljenje. Bistvo je ustrezna razdelitev nalog komponentam.

3.1.1.1 Raspberry Pi

Raspberry Pi del predstavlja krmilni sistem s senzorji in aktuatorji, ki omogočajo interakcijo računalnika z zunanjim svetom. Programski del naj bi bil sestavljen iz naslednjih delov:

- `TCPServer` - sprejem povezave, sprejem ukazov, posredovanje statusnih podatkov in posredovanje ukazov izhodnim napravam;
- `Gstreamer` - zajem in posredovanje slike, video toka preko povezave UDP;

- `Iniciative client` - vzpostavitev povezave z mobilno napravo, pošiljanje svojega naslova IP in skrb za ponovno vzpostavljanje povezave;
- `IPCarrier` - shrani IP naslov mobilne naprave, posreduje naslov IP mobilne naprave in ga preveri;
- `QRReader` - prebere QR kodo, jo preveri, posreduje naslov IP;
- `ServoHandler` - posredovanje ukazov na GPIO (ukazi servo motorjem);
- `MotorHandler` - posredovanje ukazov na GPIO (ukazi pogonskemu motorju);
- `RotationHandler` - pridobivanje nagiba iz merilnika pospeška (branje senzorjev);
- `SoftwareUpdater` - posodobi datoteke na RPi (skrbi za ažurnost kode, ko je izdelek v rokah uporabnika).

3.1.1.2 Mobilna naprava

Mobilna naprava (telefon ali tablica) omogoča nadzor strojne opreme s pomočjo nagibov naprave in kontrolnih gumbov aplikacije. Glavne komponente naj bi bile:

- `TCPClient` – vzpostavitev povezave preko TCP, posredovanje ukazov;
- `IniciativeServer` - sprejem povezave, sprejem IP-ja RPi;
- `IPCarrier` - shrani naslov IP RPi in ga posreduje ostalim komponentam;
- `ImageViewer` - prikaz video pretoka iz omrežja, ki ga pošilja RPi;
- `HUD` - obvladovanje prikaza horizonta za namen orientacije.

3.1.2 Uporabniške zgodbe

Za ustrezno definicijo končnega izdelka je potrebno razumevanje zahtev uporabnika, ki so ključen indikator uspešnosti projekta kot zahtevnost implementiranih algoritmov.

Po standardnem pristopu pri načrtovanju programske opreme bomo definirali naslednje uporabniške zgodbe (dogodke, ki jih uporabnik doživlja in uporablja med uporabo izdelka):

- uporabnik lahko gleda sliko v živo iz letala na mobitelu in katerikoli drugi napravi;
- uporabnik lahko pogleda, koliko je preostali čas leta;
- uporabnik lahko z nagibom mobilne naprave upravlja letalo;
- uporabnik se lahko orientira glede na horizont;
- uporabnik lahko ogleda video v polni kvaliteti s pomočjo posnetka na SD kartici;
- uporabnik lahko upravlja letalo, če obstaja internetni dostop;
- uporabnik nima skrbi, če ena od naprav začasno izgubi signal;
- uporabnik lahko izbira med več načini leta;
    - stabiliziran (naklon letala enak naklonu telefona);
    - akrobatski (naklon zakrilc se spreminja z nagibom telefona);
    - pameten (stabiliziran z avtopilotom) - spreminjanje parametrov višine, hitrosti, smeri, vzpona/padca in naklona.

3.1.3 Koraki izdelave

Okvirno imamo postavljene zahteve, kaj naj bi rešitev počela. Sedaj lahko napišemo predvidene korake izdelave, da se lahko sistematično lotimo načrtovanja. Definiramo naslednje korake izdelave:

- priklop RPi na omrežje preko UTP (prvi korak je spoznavanje z osnovami RPi);
- nadzor preko SSH (spoznavanje z omrežnimi protokoli);
- vzpostavitev komunikacije RPi s testno mobilno aplikacijo, prenos podatkov (komunikacija in prenos podatkov v obe smeri je ključni del našega izdelka, brez te osnove ne gre), komunikacija preko ethernet omrežja (protokol TCP, uporaba vtičnikov);
- vzpostavitev korakov inicializacije povezave;
    - mobilna aplikacija naredi strežnik, svoj (javni ali privatni) IP zapiše v QR kodo;
    - RPI s kamero prebere QR kodo in shrani naslov IP mobilne naprave;
    - RPI postane odjemalec, pošlje svoj naslov IP mobilni napravi, postavi strežnik;
    - mobilna aplikacija ustavi strežnik in postane odjemalec;

    - povezava je vzpostavljena;
    - ob izgubi povezave ene naprave, je možna ponovna vzpostavitev povezave;
- priklop kamere in prenos slike (prenos slike je ključen za našo rešitev);
- ali PiCamera ali GStreamer, več ponorov videa (v nadgradnji);
- shranjevanje videa na SD kartico (za poznejši pregled in uporabo, deljenje posnetkov);
- izdelava uporabniškega vmesnika aplikacije – njegove glavne funkcije;
    - prenos video pretoka iz kamere;
    - prikaz hitrosti glavnega motorja;
    - prikaz nagiba telefona;
- upravljanje servo motorjev (s pomočjo mobilne aplikacije);
- priklop pospeškomera in uporaba za nadzor stabilnosti;
- priklop RPi na 3G/4G omrežje in test delovanja implementiranih funkcionalnosti;
- testiranje in optimizacija latence, prenosa slike na omrežjih 4G/3G/2G/E;
- izdelava dizajna letala (s pomočjo osnovnih orodij);
- izbira in integracija komponent (uporaba rešitve v realnem svetu);
- testiranje leta (preizkus delovanja, da vidimo, kaj ne deluje in kaj je potrebno izboljšati);
- vzpostavitev varnostnih mehanizmov ob izgubi signala (varnost in legalnost je na prvem mestu).

3.1.4 Funkcionalnosti aplikacije

Razvita aplikacija naj bi omogočila naslednje funkcionalnosti:

- komunikacijo z RPi preko internetne povezave;
- telemetrijo (prikaz podatkov iz senzorjev - napetost baterije, signal);
- kontrolo servo motorjev in pogonskih elektromotorjev;
- stabilnost - nadzor naklona preko naklona pametnega telefona;
- gledanje slike v živo na zaslonu;

- prikaz horizonta in naklona na zaslonu spredaj – v ozadju se pretaka video.

3.1.5 Funkcionalnosti strojne opreme

Prav tako določimo naloge strojne opreme iz tehniškega vidika, da bomo lažje razumeli, kaj sploh načrtujemo:

- povezava na omrežje preko 3G/4G modema (za neomejen doseg);
- sprejem ukazov (izvajanje ukazov, ki jih pošilja mobilna naprava);
- prenos slike s čim manjšo latenco odjemalcu (odjemalec je mobilna naprava);
- shranjevanje videa na spominsko kartico (za kvaliteten video, ki ga ni mogoče prenesti preko omrežja).

3.1.6 Sestava strojne opreme

Preden začnemo z načrtovanjem je potrebno vedeti, kaj bo naša strojna oprema sploh vsebovala. V ta namen določimo glavne komponente strojne opreme:

- RPi + spominska kartica (hranjenje operacijskega sistema in podatkov);
- kamera PiCamera (snemanje in fotografiranje);
- 3G/4G modem (prenos podatkov po mobilnem omrežju);
- enostavno letalo;
- merilnik pospeška (senzor, potreben za branje naklona);
- ohišje (sestavni del, ki omogoča uporabo v realnih okoliščinah);
- razširitveno vezje za pogon servo motorjev (vezje za opravljanje nalog, ki jih Raspberry Pi zaradi svojih pomanjkljivosti ne more opravljati).

3.1.7 Predvidena sestava letala

Zadnji korak pred načrtovanjem korakov izdelave je postopek sestave letala. To je bolj splošni del, ki uporablja tako programsko kot strojno opremo na obeh fizičnih straneh in opravlja naloge, ki so vidne v fizičnem svetu.

- strojna oprema v ohišju (strojni del naše rešitve);
- baterija (skrb za nemoteno električno energijo);
- pogonski elektromotor (za premikanje letala);
- ESC krmilnik (za krmiljenje servo motorjev in pogonskega elektromotorja) ter
- 2 servo motorja za zakrilca (za fizično premikanje zakrilc in usmerjanje letala).

3.2 Strojna oprema rešitve

Tukaj bomo predstavili opis strojne rešitve, ki smo jo uporabljali med razvojem (nekatere tudi v končnem izdelku) za namen sprotnega testiranja brez letenja in razvoja osnovnih funkcionalnosti komponent, ki so ključne za delovanje naše rešitve. Med testne komponente sodijo servo motorji, baterija in krmilnik motorja.

3.2.1 Mikrokrmilnik in računalnik RPi 2

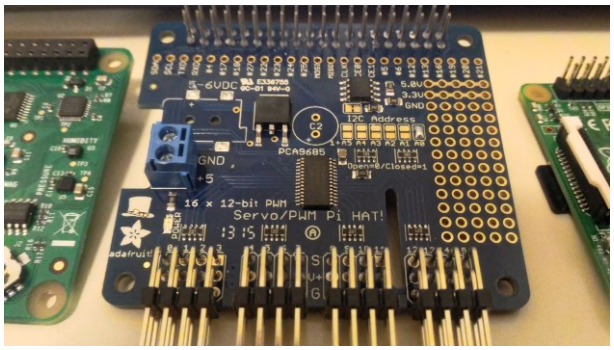
RPi 2 Model B+ [36] na sliki 7 je trenutno predzadnja verzija generacije RPi. Zmogljiv SoC združuje 900MHz 4-jedrni CPU ARM Cortex-A7, 1GB pomnilnika RAM in VideoCore IV GPU. Glavni vhodi/izhodi, pomembni na nas so pini GPIO (pri tem je pomembno I2C vodilo), vrata USB, vrata CSI za priklop podprte strojno pospešene kamere in vrata micro USB za napajanje. Ostali deli, kot so HDMI in ethernet so ključni le v zgodnjih fazah razvoja.

Slika 7: Raspberry Pi 2

3.2.2 Dodatek Servo HAT

Čeprav je RPi zmogljiv računalnik, za krmiljenje servo motorjev potrebuje dodatek (slika 8), saj poganja jedro operacijskega sistema Linux, ki ne teče v realnem času. Eden teh je 16-

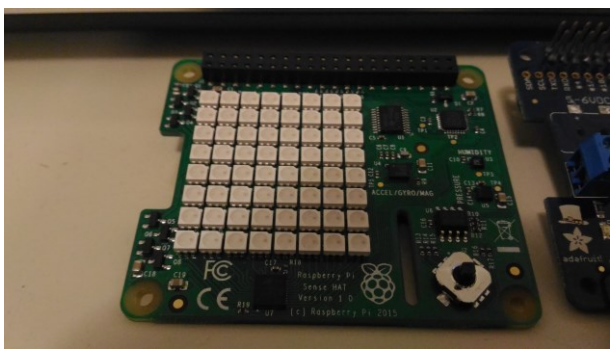
kanalni razvojni dodatek Servo HAT [37], ki omogoča natančno krmiljenje do 16 servo/brezkrtačnih motorjev. Ti namreč delujejo s pomočjo krmilnega signala PWM. Na ta način ne obremenjujemo CPU-ja na mikrokrmilniku RPi, hkrati pa dosegamo krmiljenje z do 12-bitno natančnostjo na do 992 servo motorjih (preko vodila I2C). Sam krmilni čip PCA9685 se napaja preko reguliranega 3.3V izhoda na RPi. Ker pa RPi ne ponuja vira napetosti, ki bi podpiral obremenitve nad 40mA, potrebujemo za same servo motorje zunanje napajanje.



Slika 8: Servo HAT

3.2.3 Dodatek Sense HAT

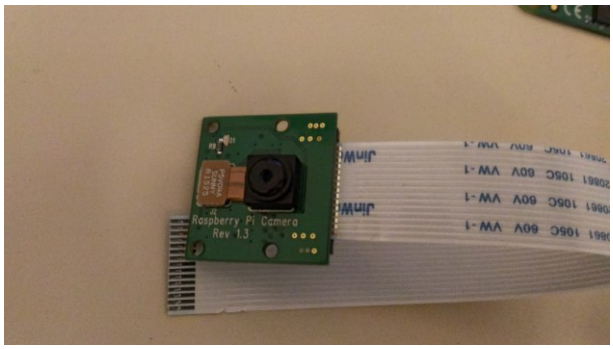
Plod projekta Astro Pi, ki je omogočal poganjanje programov učencev in dijakov iz celega sveta na mednarodni vesoljski postaji ISS, je bil dodatek Sense HAT [38], kot ga vidimo na sliki 9. Ta vsebuje žiroskop, pospeškomer, magnetomer, temperaturni, vlažnostni in tlačni senzor. Ti elementi so pomembni za delovanje naše rešitve razen matrike LED velikosti 8x8 in majhne krmilne palice. Prva je v pomoč pri izpisu lokalnega naslov IP, če nimamo monitorja, da se lahko potem preko vmesnika SSH povežemo na RPi.



Slika 9: Sense HAT

3.2.4 PiCamera

Zaznavanje pogleda iz letala omogoča video kamera PiCamera [39], ki je bila posebej razvita s strani fundacije RPi za učinkovito strojno pospešeno kodiranje, dekodiranje in obdelovanje videa (s pomočjo GPU in posebej napisanih gonilnikov). Za naš izdelek je bila uporabljena verzija V1 (slika 10), ki se od novejše verzije V2 najbolj opazno razlikuje v resoluciji 8MP, kar je 3MP več od verzije V1. Za naše potrebe je več kot dovolj že prva.



Slika 10: PiCamera

3.2.5 Mrežna kartica USB Wi-Fi

Uradna brezžična USB mrežna kartica [40] na sliki 11 je bila ključna do poznih faz razvoja, saj je omogočala brezhibno in komunikacijo brez stroškov preko omrežne dostopne točke, postavljene na usmerjevalniku ali računalniku.

Slika 11: Brezžična Wi-Fi mrežna kartica z USB priklopom

3.2.6 LTE modul

Mobilna USB mrežna kartica Huawei E3372 4G LTE [41] na sliki 12 s hitrostjo 150/50 Mb/s je ključna za povezavo RPi v internetno omrežje brez posebnih omejitev razdalj, saj obstoječa infrastruktura omogoča dokaj široko razširjeno povezljivost na 3G ali 4G omrežje. Mrežna kartica ima tudi vmesnik za uporabo spominske kartice, kar zelo olajša snemanje videa nanjo.

Slika 12: Modem 4G LTE z USB priklopom

3.2.7 GPS modul

GPS sprejemnik U-blox Neo-6M [42] na sliki 13 je v poznih fazah razvoja prispeval k zanesljivosti, udobnosti in kakovosti uporabe naše rešitve. Najbolj pomembne lastnosti so natančnost do 3m, čas do prve povezavo pod 30s, osvežitev podatkov frekvence 5hHz, natančnost hitrosti glede na zemljo 0.1m/s, široka podpora različnih standardov satelitov, protokolov (NTP, PPP) in do 50 kanalov spremljanja ter 20 kanalov uporabe. Število kanalov pove, koliko podatkov različnih satelitov lahko GPS sprejemnik sledi.

Slika 13: Sprejemnik GPS ublox Neo-6M

3.2.8 Napajanje - napajalnik, baterija

Za napajanje RPi potrebujemo napajalnik [44] s priklopom micro USB, ki zmore tok 1-2A (odvisno od porabnikov). Priklop na priključek microUSB omogoča varno uporaba RPi, saj tok teče preko varovalke in na različne napetostne tokokroge. Za testiranje je napajalnik dovolj, za mobilno uporabo pa je potrebno poskrbeti napajanje iz baterije primerne vrste, napetosti in toka. Možnosti je več, zaradi enostavnosti uporabimo ustrezno LiPo baterijo, ki jo prikazuje slika 14, ki lahko hkrati poganja pogonski elektromotor in servo motorje, pa tudi RPi, vse iz enega vira.

Slika 14: Baterija LiPo

3.2.9 Kartica SD

Spominska kartica je oprema, brez katere RPi ne deluje, saj iz nje naloži operacijski sistem, hkrati pa jo uporablja kot spominski medij. Priporočljiva je velikost večja od 4GB, odvisno pa seveda od namena uporabe. Velikost nenameščenega celotnega operacijskega sistema Raspbian znaša 1.3GB, minimalna verzija pa znaša pod 300MB prostora. Pri prevajanju programske opreme lahko potrebujemo tudi več GB prostora, zato smo za naš namen izbrali dovolj velikost 16GB.

3.2.10 Dodatna strojna oprema testnega okolja

V tem poglavju bomo predstavili strojno opremo, ki ni del naše osnovne rešitve, vendar smo jo uporabljali za razvoj rešitve zaradi uporabe v končnem izdelku.

3.2.10.1 Servo motorji

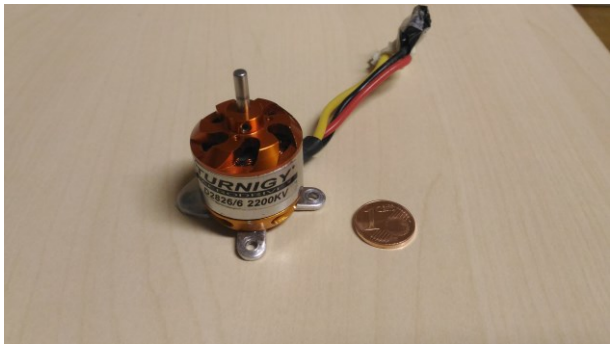
Servo motor je obračalni aktuator, ki za razliko od bolj poznanih splošnih elektromotorjev omogoča natančno nastavitev kotne pozicije. Poznamo analogne in digitalne servo motorje, razlika je v načinu procesiranja signalov. Analogni delujejo direktno na PWM signal, digitalni pa vsebujejo krmilno vezje s hitrim procesiranjem, ki lahko več kot 5-kratno povečajo

frekvenco. Rezultat so večja natančnost, boljši odziv in navori, hitrejši pospeški, a tudi večja poraba energije (več 100mA). Pri tehtanju prednosti in slabosti za naš primer potrebujemo digitalne, saj poraba energije v primerjavi s porabo pogonskega elektromotorja ni pomembna. Slika 15 prikazuje uporabljena servo motorja.

Slika 15: Servo motorja

3.2.10.2 Brezkrtačni motor

Za pogon letala uporabimo brezkrtačni elektromotor moči 342W, ki z ustreznimi propelerji proizvede do okoli 1kg potiska. Sam motor tehta 50g. Elektromotor deluje na napetosti 11V, kar pri specifikaciji 2200Kv (obratov na volt/minuto) pomeni 24200 obratov/minuto pri neobremenjenem vrtenju. Hitrost vrtenja je pri pravih letalih skupaj s premerom propelerja pomembna zaradi hitrosti vrtenja na konicah propelerja, ki v primeru, da presega hitrost zvoka (en Mach), izgublja na učinkovitosti.



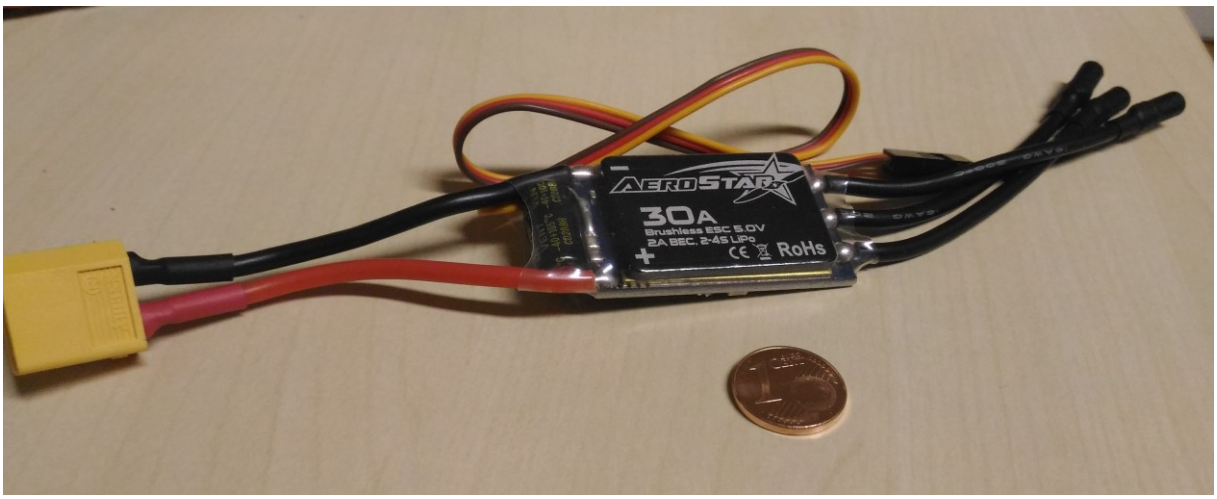
Slika 16: Pogonski brezkrtačni elektromotor Turnigy D2826/6

3.2.10.3 Krmilnik ESC

Za krmiljenje brezkrtačnega motorja potrebujemo še poseben krmilnik (slika 17). Ta vsebuje vezje, ki iz ločenega tokokroga napaja motor na napetosti (tipično 14, lahko tudi do 30V+), ki bi mikrokrmilnik uničila. Sam način uporabe je identičen krmiljenju servo motorjev, glavna

razlika vidna uporabniku je varnostna inicializacija, ki preprečuje nenamerne poškodbe zaradi človeške nepazljivosti.

Vezij za krmiljenje brezkrtačnih motorjev je več vrst. Nekateri ponujajo tudi napajanje za dodatno opremo, drugi pa ne [43]. V splošnem se krmilnik imenuje ESC, ki združuje tudi dodatno vezje, ki skrbi za napajanje morebitnih dodatnih naprav.



Slika 17: Krmilnik ESC

3.3 Programska rešitev na mikrokrmilnem sistemu RPi

Pri razvoju programske rešitve na RPi smo najprej morali postaviti ustrezno delovno okolje, da smo lahko dejansko začeli z delom. Najprej bomo opisali začetne korake, sledi podrobnejši, a jedrnat in skrčen opis postopka raziskovanja in razvoja skozi težave in delne rešitve, na koncu pa bomo predstavili arhitekturo končnega sistema, ki ustreza zasnovanemu delovanju.

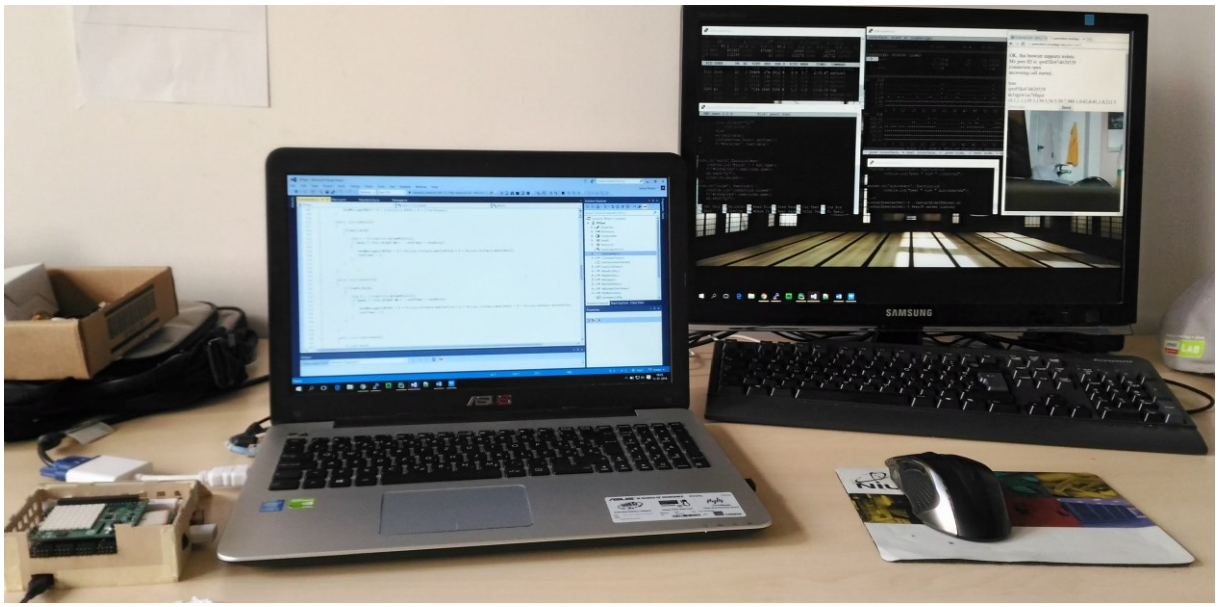
3.3.1.1 Vzpostavitev delovnega okolja

Za vzpostavitev osnovnega delovnega okolja (slika 18) je bilo potrebno kar nekaj korakov, saj smo hoteli robustno arhitekturo, ki ni odvisna od končne arhitekture. To pomeni, da lahko razvijamo stvari kljub temu, da nimamo priklopljenega RPi na monitor, tipkovnico, miško in ethernet kabel oziroma mikrokrmilnika sploh nimamo zraven. Za določene stvari je bil dovolj računalnik in testiranje Python programov kar lokalno, kar je omogočalo prenosljivost in lažje testiranje.

Končna konfiguracija je izgledala tako: RPi je povezan v električno omrežje z adapterjem, z brezžično USB mrežno kartico je priklopljen na omrežje preko dostopne točke, vzpostavljene na računalniku s programsko opremo mHotspot, ki omogoča vzpostavitev brezžične dostopne točke. Do njega dostopamo s povezavo protokola SSH v programu Putty ali podobnim, ker pa

RPi lahko poganja VNC (angl. Virtual Network Computing) [45] servis, tudi z VNC odjemalcem preko X11 tehnologije [46] direktno do grafičnega vmesnika imenovanega X [46].

Ker je programiranje Python programa potekalo na PC-ju v PyCharm integriranem razvojnem okolju [130], je bilo v poznih fazah razvoja potrebno prenašati kodo na RPi. V ta namen je bil ustvarjen Git repozitorij, preko katerega smo s PyCharmovo Git podporo neposredno nalagali, ter s svojo preprosto bash skripto posodabljali kodo na RPi. Ostalih bolj nerodnih metod se nismo hoteli posluževati (kot je recimo pošiljanje preko FTP strežnika ali celo fizičnega kopiranja na SD kartico).



Slika 18: Delovno okolje

3.3.1.2 Raziskovanje in razvoj

Prvotni program je deloval sekvenčno, saj ni bilo potrebe za paralelizacijo, po principu YAGNI [47] (princip metodologije ekstremnega programiranja, pri čemer ne razvijamo nečesa, kar ne bomo potrebovali). Težili smo k enostavnosti programa, dokler se ni izkazalo, da potrebujemo nekaj drugačnega.

Sekvenčni program

Sekvenčni program je vseboval komponente `sensors` (branje senzorjev), `comm` (komunikacija), `commander` (izvajanje), `main` (glavni program) ter pomožne komponente. Vse je bilo modularno in je delovalo odlično, dokler se ni izkazalo, da zaradi knjižnice Selenium sekvenčni program ni več ustrezen, ker ustavlja ostale dele programa, medtem ko komunicira z brskalnikom. Sledila je prvo prestrukturiranje tega dela rešitve.

Nitenje, GIL in multiprocesiranje

Logična izbira je bilo nitenje. Potrebovali smo ga za ločitev branja senzorjev, ki zaradi čisto fizikalnih razlogov (to je zahtevanega visokofrekvenčnega vzorčenja) potrebuje stalno izvajanje brez večjih premorov. To je sicer izboljšalo stvari, vendar ni bilo dovolj. Visoka obremenitev knjižnice Selenium, brskalnika in pretok videa je premor med dvema poslanima ukazoma trajal nad 200ms, zamik videa pa tudi do 700ms. To je za zahteve realnega časa v primeru naše rešitve odločno preveč. Raziskovanje nas je privedlo do uporabe niti. Hoteli smo ločiti branje, pošiljanje podatkov in video pretok, torej iz enega dela v tri dele, vendar smo po testiranju in branju literature ugotovili, da ima Python posebne funkcionalnosti v primeru nitenja. V primeru dela nad istim objektom Python uporablja GIL (Global Interpreter Lock [48]). To je ključavnica, ki skrbi, da ne pride do nekonsistentnosti podatkov. Redna praksa pri uporabi paralelizacije z nitmi je namreč zaklepanje kode, da pri spreminjanju objekta ne pride do sočasnega branja s strani druge niti. Python za to poskrbi sam, slaba stran tega pa je, da ne pride do prave paralelizacije, saj nimamo dostopa do ročnega zaklepanja.

Nitenje je vseeno uporabna stvar tudi v Pythonu, ampak samo, če gre za I/O operacije, da medtem med prenosom datotek iz interneta glavni program ne stoji. V primeru operacij, vezanih na CPU, pa je nitenje manj uporabno. Multiprocesiranje [49] je bil majhen del poskusnega razvoja, saj bi morali uporabljati globalne spremenljivke za naš specifičen problem. Komunikacija med procesi namreč poteka enosmerno preko vrst ali dvosmerno preko cevovodov, mogoč pa je prenos le primitivnih podatkovnih virov, kar v primeru takratnega sekvenčnega programa ni bilo mogoče. Sicer obstajajo možnosti za serializacijo, vendar bi morali vsak tip objekta, ki ga razred vsebuje, ročno serializirati, ker Pythonov vmesnik za serializacijo Pickle [50] ali kakšenkoli drug vmesnik zaradi čisto teoretičnega ozadja tega ne zmore sam.

Delitev na več procesov

Naslednja ideja je bila ločiti kritično področje na 2 dela in tako razbremeniti RPi, saj je brskalnik Firefox porabil 100% enega procesorja, ni pa očitno zgrajen za paralelno izvajanje v enem procesu. Zato smo ločili prenos podatkov in video v 2 ločena procesa in ločili krmiljenje s knjižnico Selenium na 2 dela. Pokazali so se dobri učinki izboljšave, saj smo dosegli 50ms zamika med pošiljanjem ukazov, ko se video ni predvajal. To je 20Hz, kar je že zelo dobro v primerjavi s prejšnjimi 7Hz. Žal pa je nadgradnja brskalnika kljub poskusom ustvarjanja premorov med pošiljanji podrla vse, saj je to upočasnilo preostali program. Pri uporabi video pretoka se je zamik med ukazi povečal na 300ms, kar je bilo nesprejemljivo. Videli smo, da to ne bo ustrezen način.

Spletni vtičnik (angl. websocket)

V intervalih je sledilo možgansko viharjenje (angl. brainstorming), raziskovanje in razvoj manjših programov in integracija v obstoječ program. Brainstorming o nitenju, multiprocesiranju in raziskovanje nas je privedlo do spoznanja, da bo potrebno za uporabno in

robustno rešitev napisati svoj način komunikacije med brskalnikom in Python programom. Kljub vsemu je uporaba knjižnice za testiranje spletnih aplikacij za hitro komunikacijo med Javascriptom in Pythonom neustrezna in popolno pretiravanje, tako zaradi količine komunikacije kot tudi porabe virov računalnika, še najbolj procesorja. Ideje so vodile k TCP-ju, vendar Javascript ne more direktno komunicirati po protokolu TCP, ampak s spletnimi vtičniki, kar je bilo za nas čisto novo odkritje. Imeli smo dve možnosti, in sicer odločiti se za Python knjižnico, ki bo podpirala povezavo z spletnimi vtičniki, ali za Javascript knjižnico, ki bo podpirala TCP povezavo. Pregledovanje obstoječih rešitev nas je pripeljalo do kratkega testiranja učinkovite rešitve s pomočjo knjižnic:

- Autobahn [51]
- Easywebsocket [52]
- Websockify [53]
- Simplewebsocket [54]
- Tornado [55]

Vmes smo izmerili čas zmogljivosti izvajanja brskalnika, Javascripta in WebRTC-ja pri pošiljanju podatkov po protokolu WebRTC. Na prenosniku z 3.0HGHz i7 procesorjem in 8GB pomnilnika smo izmerili 0.5ms časa izvajanja ukaza za pošiljanje, na RPI pa 5ms. To je bila motivacija in odkritje, da je možno rešitev narediti bolje in bolj učinkovito, predvsem pa iz stališča pošiljanja hitreje. Odločili smo se za tretjo možnost. Postaviti je bilo potrebno ustrezen strežnik spletnih vtičnikov, s katerim bosta komunicirala tako Python in Javascript odjemalca spletnih vtičnikov. Zaradi preveč ali premalo robustnih ali ne vzdrževanih knjižnic, neuspešnih implementacij in strme krivulje učenja ostalih orodij smo izbrali Tornado - robusten in enostaven strežnik spletnih vtičnikov. Implementirali smo ga v Pythonu. Po razvoju majhnega programa za preverjanje ustreznosti za našo rešitev je bilo potrebno implementirati strežnik tako, da deluje kot posrednik oziroma glede na identifikacijski znak pošlje sporočilo ustreznemu odjemalcu.

Večprocesorski program, IPC in ZeroMQ

Še vedno pa je bilo potrebno odpraviti posebno slabost - sekvenčnost programa, ki je nista rešila niti nitenje niti multiprocesiranje. Rešila nas je knjižnica Zero-MQ [56] (od tu izhaja tudi poimenovanje mnogih komponent), ki rešuje problem hitre in učinkovite medprocesne komunikacije [57] na različnih operacijskih sistemih in v različnih programskih jezikih. Za našo rešitev je bil idealen sporočevalni vzorec objava – naročilo (angl. publish-subscribe) [58], saj ne povzroča dodatnega presežka komunikacije (angl. overheada). Potrebna je enosmerna komunikacija in ni pomena, da vsakič zahtevamo podatek, kot to počne recimo vzorec zahteva - odgovor (angl. request-reply), če podatek pričakujemo. Poleg tega naročniki (angl. subscribers) lahko s pomočjo teme (angl. topic) izločajo sporočila, ki jim niso namenjena.

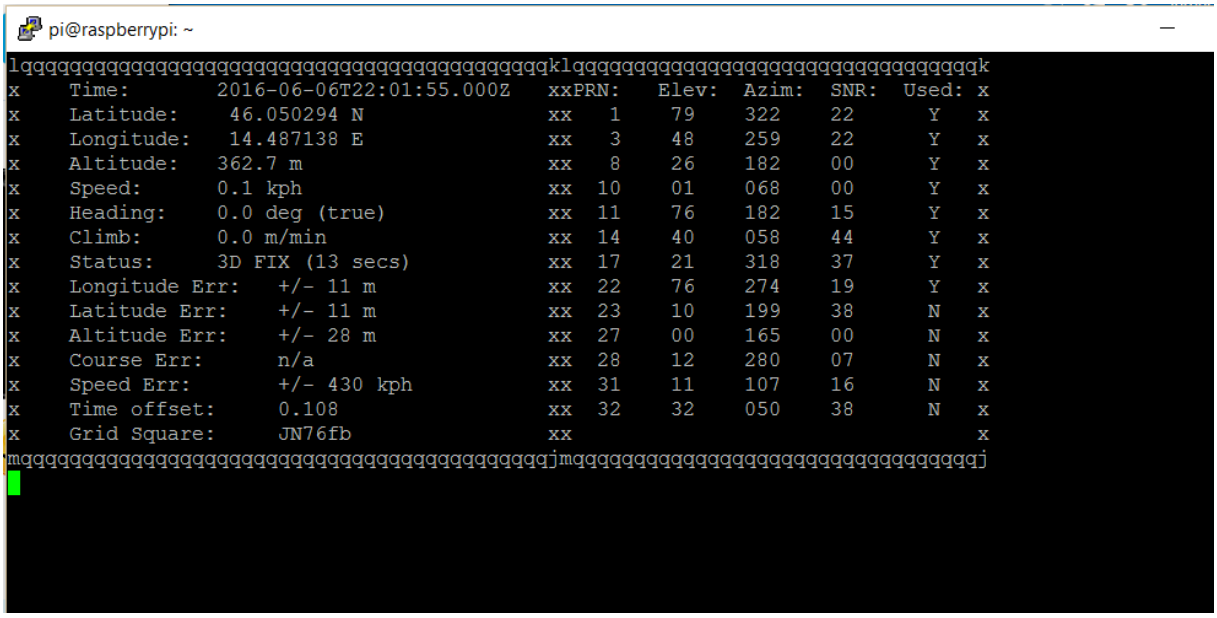
Pomembne lastnosti knjižnice ZMQ so:

- hitrost, enostavnost (8M sporočil/sekundo, 30 nanosekund latence) [59];
- inteligentna knjižnica za sporočanje;
- povezava in pošiljanje na več vtičnikov (angl. socketov);
- nitenje;
- integrirani sporočevalni vzorci in strukture, najbolj enostavni od teh so:
    - zahteva-odgovor (angl. requst-reply);
    - objava-naročanje (angl. publish-subscribe);
    - potisk-vlek (angl. push-pull);
- močni usmerjevalni algoritmi in podporne strukture;
- podpora za ogromno platform in jezikov;
- odprtokodnost;
- enostavnost za uporabo;
- uporabniki: AT&T, CERN, Cisco, EA, Microsoft, NASA, Samsung, itd.

Integracija GPS

V kasnejših fazah razvoja smo v obstoječo rešitev dodali GPS modul. Najprej smo ga povezali preko USB - FTDI vmesnika in ga nastavili v okolju Windows s pomočjo uradne programske opreme u-blox u-center v8.21 [138]. Ta nam omogoča povezavo preko COM vrat, prikaz prejetih podatkov, konfiguracijo in veliko več. Nekateri pomembnejši parametri pri konfiguraciji so `ANT` (antena), `CGF` (konfiguracija), `NAV5` (model navigacije), `NMEA` (verzija protokola, ki določa strukturo GPS podatkov), `PMS` (upravljanje z energijo), `PRT` (vrata in pasovna širina), `RATE` (frekvenca osveževanja in vir ure), `TXSLOT` (vmesniki za komunikacijo), `USB` (nastavitve USB)

Nato smo ga preko USB vmesnika povezali na RPi in namestili ustrezno programsko opremo GPSd [61], pri čemer je potrebno tudi v Linuxu serijskemu vmesniku posebej povedati, katero pasovno širino naj uporablja. Delovanje smo preverjali v ukazni vrstici (slika 19).



Slika 19: Prikaz podatkov GPS sprejemnika

Za uporabo v Python knjižnici smo morali namestiti ustrezen Python vmesnik [62]. Ko smo imeli delujoč primer, smo ga vstaili v obstoječ sistem in dodali nekaj stvari v naš komunikacijski protokol. Kasneje je komunikacija GPS - RPi potekala preko serijskega vmesnika. Mogoče kot zanimivost, ker se pri medprocesni komunikaciji v Pythonu v naši implementaciji dodaja znake za temo in kasneje briše, da dobimo originalno sporočilo, je prišlo do zanimivega hrošča, ko smo pri uporabi številčnih tem pri brisanju s funkcijo pobrisali tudi del koordinate in pri tem se naj bi nahajali nekje v Afriki.

V poznih fazah razvoja smo kupili še ustrezen 4G modem, namestili ustrezno programsko opremo Sakis3G [63], ki omogoča samodejno konfiguracijo 3G/4G modemov na Linux sistemih. Slabost kupljenega modema Huawei e3372h se je izkazalo intenzivno gretje same naprave.

3.3.1.3 Arhitektura – glavne komponente

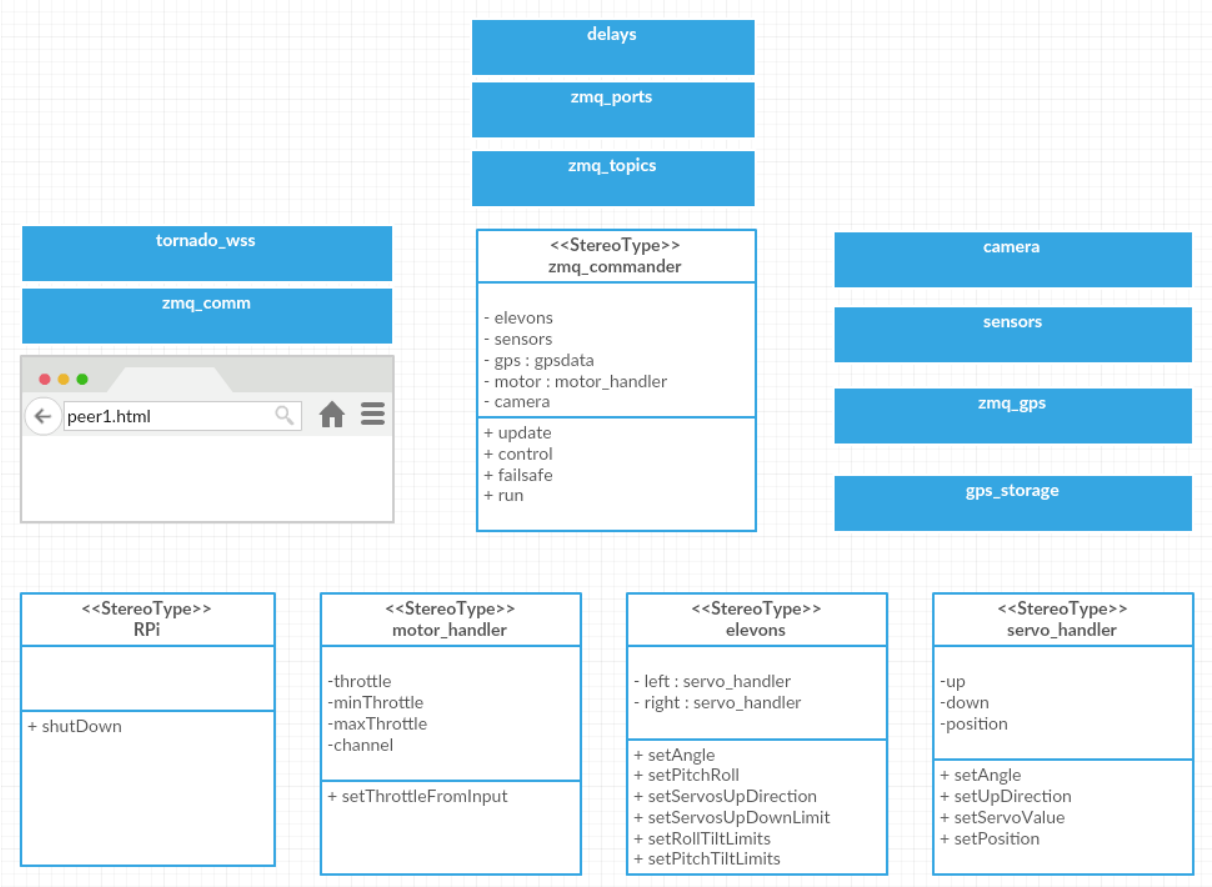
Glavne komponente končne rešitve (slika 20) na mikrokrmilniku RPi:

- `zmq_main` nastavi video parametre s pomočjo vmesnika `v4l2-ctl` [150], virtualni zaslon preko Xvfb zaslonskega strežnika in preko njega požene brskalnik Firefox in odpre spletno komunikacijsko aplikacijsko stran in požene Python skripte s primerno verzijo Pythona, po potrebi z ukazom za administrativne pravice;

- `zmq_sensors` - skrbi za komunikacijo z dodatkom Sense HAT, branje in shranjevanje podatkov o senzorjih (nagibi, smer, temperatura, vlažnost, tlak, pospeški, preračunana višina iz tlaka);

- `zmq_GPS` - uporablja Python knjižnico GPSd in skrbi za komunikacijo z GPS modulom in branje GPS podatkov (geografska dolžina in širina, višina, hitrost, dvig/spust, smer);
- `GPS_storage` skrbi za shranjevanje GPS podatkov in posredovanje teh na zahtevo;
- `tornado_wss` je strežnik spletnih vtičnikov, ki skrbi za posredovanje sporočil med brskalniškim Javascriptom in komunikatorjem preko povezave spletnih vtičnikov;
- `zmq_comm` je komunikator, ki skrbi za posredovanje sporočil med `tornado_wss` (in posledično brskalniškim javascriptom) preko povezave spletnih vtičnikov in IPC preko povezav TCP. Preko IPC je povezan tudi na `zmq_GPS` in `zmq_commander`;
- `zmq_commander` predstavlja izvajalni razred, ki skrbi za izvajanje ukazov in funkcij in ima popoln nadzor nad dogajanjem. Preko IPC je povezan na `zmq_comm`, `zmq_sensors`, `zmq_GPS`. Prav tako ima s pomočjo kompozicije nadzor nad kamero, servo dodatkom (in s tem nadzor nad servo in pogonskimi motorji), hrani pa tudi GPS podatke in uporabniške podatke, ki so nastavljeni parametri za stabilizacijsko in avtopilot funkcijo;
- `zmq_topics` vsebuje konstante z IPC temami;
- `zmq_ports` vsebuje konstante z IPC vrati;
- `delays` vsebuje konstante z zamiki osveževanja raznih podatkov;
- `camera` skrbi za operacije nad kamero (ukazi za fotografiranje/snemanje);
- `RPi` skrbi za varno in hitro zaustavitev sistema RPi na daljavo;
- `servo_handler` skrbi za popoln nadzor nad enim servo motorjem s tehnikami omejevanja, preslikovanja vrednosti, nastavlja parametrov (pozicije, omejitve, občutljivosti in orientacije premika);
- `motor_handler` skrbi za popoln nadzor nad enim pogonskim motorjem, osnovne funkcije in pametno delovanje;
- `elevons` je razred, specifičen za leteča krila, uporablja `servo_handler` in določa funkcionalnosti za levo in desno zakrilce, njune nastavitve in pametne algoritme. Je uporabniški vmesnik za servo motorje.

Arhitektura je zgrajena tako, da vsaka komponenta opravlja svoje delo. Program teče v več procesih, saj je bilo to sprva potrebno zaradi slabe zmogljivosti knjižnice Selenium, izboljšuje dobro prakso in povečuje učinkovitost, sama rešitev pa izkorišča tudi večprocesorske sisteme.



Slika 20: Poenostavljen prikaz arhitekture in njene modularne zgradbe

3.4 Programska rešitev na nadzornem sistemu/aplikaciji Android

V naslednjem delu bomo predstavili potek in načrtovanje uporabniškega vmesnika, sledi podrobnejši opis glavnih korakov pri raziskovanju in razvoju. Za tem bomo predstavili arhitekturo programske rešitve tega dela ter funkcionalnosti sistema, ker so drugačne oziroma spremenjene (in dodane) od tistih, ki smo jih prvotno zasnovali.

3.4.1 Načrtovanje uporabniškega vmesnika

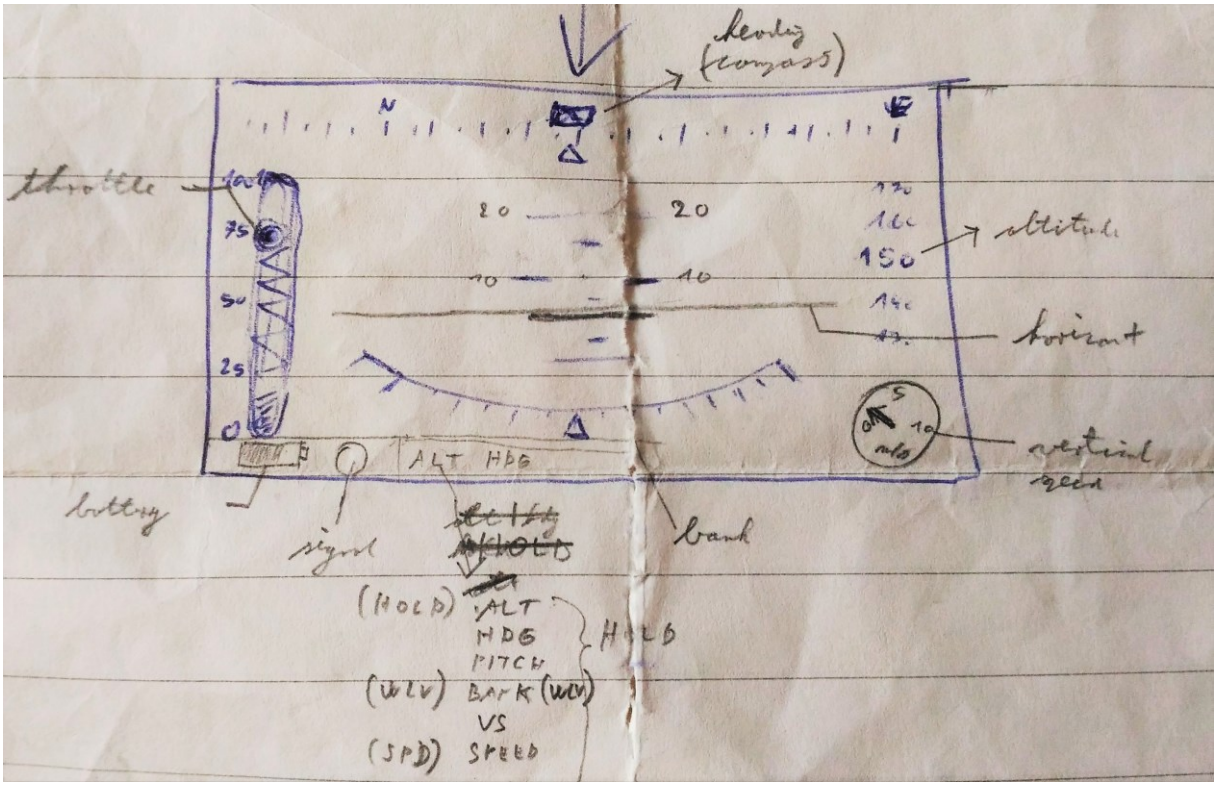
Prvi osnutki vključujejo načrtovanje osnovnega poteka dogodkov (angl. wireframing). Nekaj ključnih besed za vsak korak ter slika uporabniškega vmesnika na papirju (slika 21) je dovolj za začetek razumevanja, kaj bi uporabnik želel oziroma pričakoval od aplikacije, katere

informacije ga zanimajo, kaj se bo dogajalo in kako si sledijo dogodki. Na tem koraku se še ne obremenjujemo s podrobnostmi.

S pomočjo enostavne postavitve elementov ter prakse, kako uporabniki držijo mobilno napravo večino časa za določen primer uporabe [64], glede na namen [65], si lahko pomagamo, da bodo uporabniki naše ciljne skupine enostavno dostopali do kontrolnih grafičnih elementov na vmesniku mobitela [66] in tablice [67], saj tako izboljšamo udobje same uporabe, njeno intuitivnost in prijetno uporabniško izkušnjo.

Čeprav so prvi koraki najbolj pomembni, ni nič narobe, če se osnovni načrt spremeni, saj med razvojem, analizo in testiranjem sproti odkrijemo pomanjkljivosti in včasih tudi neprimernosti. Za primerno kakovost razvoja je boljše prilagajati osnovni načrt (če si izposodimo to smernico iz agilnih metodologij) kot da se držimo starih prepričanj ali celo konservativnih načrtov, v katere smo bili dokaj prepričani. Razvoj je namreč živ sistem in zato se zahteve ali spreminjajo ali pa ugotovimo, da smo jih "samo" napačno razumeli.

Grafični elementi za Android omogočajo kar nekaj izbire, kljub temu pa moramo za resnejšo uporabo marsikaj spremeniti. To je uporabiti zunanje knjižnice, ali pa sami implementirati in povoziti metode, ki jih potrebujemo z drugimi parametri.



Slika 21: Prvi osnutek uporabniškega vmesnika

3.4.2 Raziskovanje in razvoj

Razvoj Android aplikacije je enako kot doslej potekal neodvisno, a vseeno povezano z napredkom ostalih delov. Prvi koraki so bili učenje osnov razvoja v okolju Xamarin Android [68] in postavljanje grafičnih elementov, njihova rotacija in premiki. Za namen sprotnega testiranja delovanja smo uporabljali kar senzorje na mobilni napravi, ki smo jih potrebovali tudi pozneje. Za ta namen se je razvil neodvisen del branja senzorjev, ki smo ga kasneje integrirali v lastni servis `TiltService`. Za prikaz naklonov pa 2 smereh (angl. pitch, roll) smo uporabili `scrollView` in 2 `imageView`, pozneje pa `NumberPicker` elemente. Ker v začetku nismo našli primernih grafičnih elementov za naš namen, smo začasno ustvarili sami v orodju Gimp [137]. `Seekbar` element, ki v rešitvi služi nadzoru pogonskega elektromotorja, je služil za simuliranje kompasa in naklonov.

Sledila je postavitev ozadja in `WebView` elementa v času razvoju komunikacijskega dela rešitve. Potrebno je bilo razviti del komunikacije tudi na Android strani. Tako je lahko potekala komunikacija med Javascriptom spletnega odjemalca in našim programom. Uporabili smo Javascript vmesnik in naredili svoj razred `MyJSInterface`, ki je skrbel za ta del komunikacije, in razred `Message`, ki je skrbel za shranjevanje podatkov v enostavni obliki. Ko smo usposobili našo senzorsko ploščo na RPi do te mere, da smo lahko brali in pošiljali podatke vsaj preko lokalne povezave, smo lahko preverili tudi delovanje naših postavljenih grafičnih elementov.

Pokazalo se je, da imajo izhodne vrednosti senzorjev na Android sistemu in senzorične plošče Sense HAT različne obsege in oblike, zato smo s pretvorbo določenih spremenljivk dosegli skladnost obeh strani. Zgradili smo preprost algoritem za računanje zamika med ukazi ter postavili grafične elemente za prikazovanje ostalih senzorjev (temperature, smeri, vlažnosti in tlaka, stanje napolnjenosti baterije) in njim primerne simbole. Stanje napolnjenosti baterije še ni implementirana funkcionalnost, ker vezje po meri, ki bi merilo tako napetost kot električni tok, še ni izdelano. Prvi prototip uporabniškega vmesnika je počasi nastajal po zgledu osnutka na sliki 21. Gumbe za rešitev v sili, vklop držanja višine, smeri in gumb avtopilot so bili t.i. neaktivni (angl. dummy) gumbi, ki niso še nič pametnega počeli.

Na podlagi vmesnika smo si lažje predstavljali celotno aplikacijo, njene zmožnosti in nadaljnji razvoj. Ugotovili smo, da bi bilo smiselno imeti funkcionalnost za fotografiranje in snemanje videa. V poznejših fazah razvoja (časovno gledano) smo bolj specifično določili načine letenja, katerih princip pa se je večkrat spremenil, domnevno zaradi pomanjkanja razumevanja in razlike med končnim uporabnikom in letalnimi standardi.

Za uporabnikov pogled smo izluščili tri načine letenja:

- ročni ali akrobatski - uporabnikov vhod direktno krmili zakrilca brez popravkov;
- stabilizacijski ali pametni - naklon telefona je enak naklonu letala;

- avtopilot - nastavljanje parametrov kot so smer, naklon in višina z gumbi, pri čemer letalo samo skrbi za doseganje ciljev in premagovanje razlik med dejanskim in željenim stanjem;

Rešilni gumb v primeru zahteve uporabnika, ki je izgubil nadzor nad letalom, sproži poravnavo letala v vodoravno lego.

Na vrsti je bil razred `Commander`, ki smo ga izdelali zato, da lahko ukaze na RPi pošiljamo s preprostimi klici metod. Ta skupaj s `Commanderjem` na RPi strani uporablja isti komunikacijski protokol (oznake in njihov pomen), da je implementacija komunikacije usklajena in enostavna, hkrati pa je lažje ugotavljanje kakršnihkoli napak na obeh straneh.

Med drugim je bilo potrebno dodati še nekaj malenkosti. To je začetni zaslon (angl. splashscreen), da medtem, ko je aplikacija v zagonu, uporabnik nima neprijetnega občutka, da aplikacijo deluje počasi ali nepravilno.

Ko je bil aktuatorski del razvit za namen testiranja, se je v ta namen razvil nov podprogram `Settings`, ki je omogočal premikanje in nastavljanje občutljivosti in mej servo motorjev in območja ukaznega naklona mobilnega telefona. Poskrbeti je bilo potrebno tudi za ustrezen življenjski cikel elementa `WebView`, da je komunikacijski del neomejeno deloval ne glede na to, v katerem podprogramu se uporabnik nahaja. Ko smo določene dele na obeh straneh razvili, smo lahko odpravili simulacije večine sklopov in preverili celotno delovanje sistema.

Proti koncu razvoju trenutnega prototipa, ko smo na RPi strani dodali GPS sprejemnik, smo dodali potreben protokol tudi na Android strani in sedaj prikazovali tudi podatek o hitrosti na uporabniškem vmesniku. Nekaj razmišljanja pa nas je vodilo do ugotovitve, da če uporabnik rešitev uporablja na tak način in v takem okolju, da ne ve točno, kje se njegovo letalo nahaja, bi cenil to informacijo kljub sliki v živo, ki mu v kombinaciji s podatkom o smeri in višini ne podaja dovolj natančne informacije o samem položaju. V ta namen je raziskovanje vodilo do vmesnika Google Maps API [69], ki nam omogoča uporabo robustne Googlove storitve. Potrebno se je registrirati in v aplikacijo dodati unikaten API ključ, ki nam omogoča uporabo njihovih storitev, in to do določene meje brezplačno.

Dodali smo nov podprogram `MapActivity`, ki skrbi za prikaz zemljevida, naše lokacije in lokacije RPi. V začetkih je bil izziv že vzpostavitev same storitve. Kmalu pa smo risali markerje, poti in računali in prikazovali relevantne podatke o razdaljah. Ta podprogram bo služil avtonomnemu letu po predhodno nastavljeni poti. Poleg tega bo preko servisa dviga (angl. elevation) preverjal nadmorsko višino terena pred in pod sabo in uporabniku omogočal koristne informacije, ki so pomembne za varnost letenja.

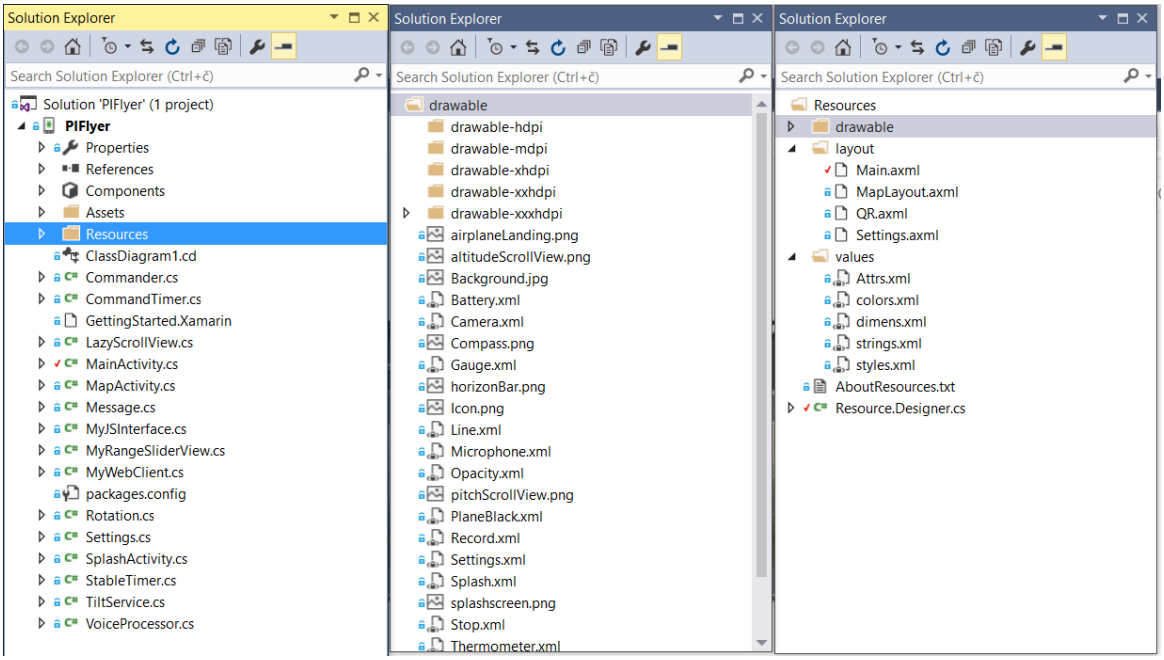
Dodali smo tudi zanimivo funkcionalnost - komponento `VoiceProcessor`, ki uporablja Androidovi storitvi za pretvorbo besedila v govor in obratno, in izvaja ukaze v načinu avtopilota, če želimo nastaviti višino ali smer (oziroma če nas zanimajo okolijski podatki, ki jih meri RPi).

3.4.3 Arhitektura

Android je zasnovan podobno kot Windows Phone, od nas namreč zahteva ločen uporabniški vmesnik od podatkov in programske logike.

Predstavili bomo glavne komponente sistema (prikazane tudi na sliki 22):

- `MainActivity` je glavni program (programsko ozadje) za glavno grafično postavitev (`main.xaml`);
- `Myreceviver` (`broadcast receiver`) je naročnik na broadcast sporočila določenih tipov;
- `Commander` predstavlja skupek metod s skupnim ukazno-komunikacijskim protokolom za pošiljanje ukazov na RPi;
- `LazyScrollView` - razred po meri - prirejen `scrollView`;
- `MapActivity` - drugi del glavnega programa, ki skrbi za prikaz lokacije RPi in je uporabniški vmesnik za načrtovanje misij in avtonomen let nasploh;
- `Message` - podatkovni singleton razred, ki sprejme, obdela in hrani prejete podatke, ki so zatem na voljo celotni aplikaciji;
- `MyJSInterface` - Javascript vmesnik za komunikacijo z brskalnikom (`WebView`) ;
- `MyRangeSliderView` - grafični element po meri, s katerim lahko določamo obseg med dvema vrednostma;
- `MyWebViewOdjemalec` - `WebView` odjemalec, ki je minimalno spremenjen za pravilno delovanje programa;
- `Rotation` - enostaven pripomoček za upravljanje z obračanjem grafičnega elementa;
- `Settings` - tretji del glavnega programa, ki skrbi na nastavitve servo motorjev in zaustavitev RPi;
- `SplashActivity` – programsko ozadje začetnega zaslona, ko se aplikacija nalaga;
- `StableTimer` - časovnik po meri za namen odštevanja reševalno-stabilizacijskega dialoga;
- `TiltService` - servis, ki bere podatke senzorjev mobitela (merilnik pospeška, žiroskop), preračuna naklone in jih shrani v razred `Message`;
- `VoiceProcessor` - skrbi za pravilen ukrep ob prepoznavanju govora.



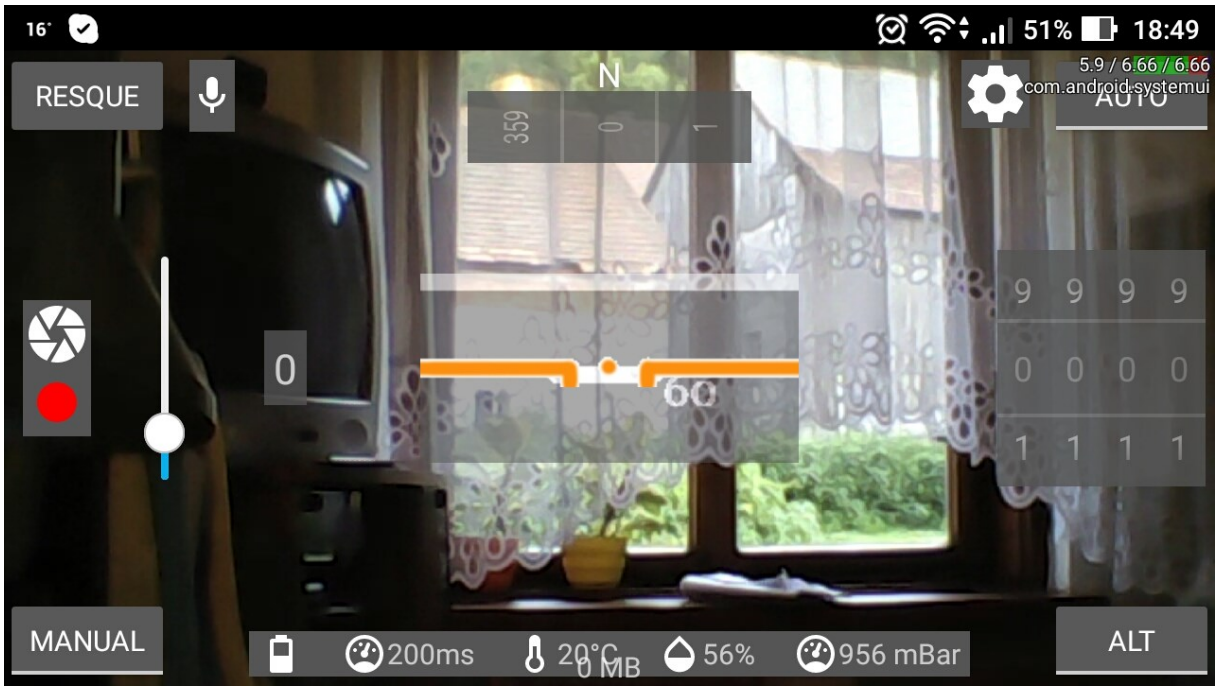
Slika 22: Prikaz strukture programske rešitve za sistem Android

3.4.4 Funkcionalnosti aplikacije

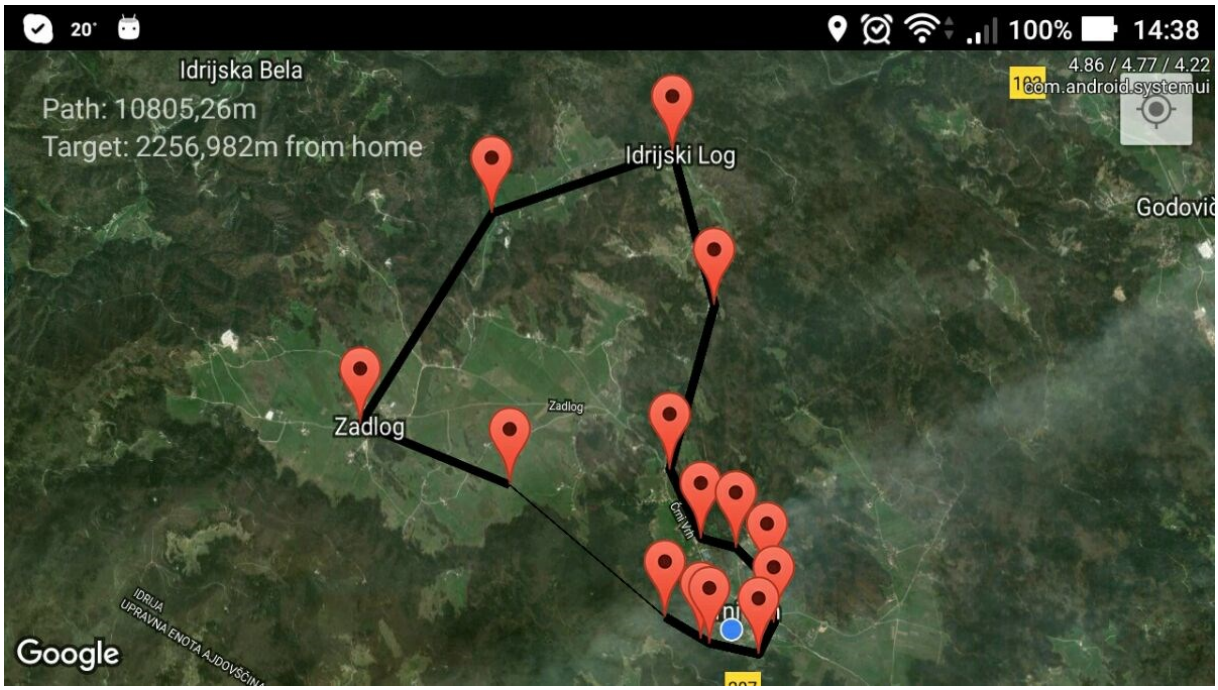
Glavne funkcionalnosti končne aplikacije so naslednje:

- prikaz podatkov v ustreznih grafičnih elementih na uporabniškem vmesniku (slika 23) podatkov RPi: video pretok, zamik med ukazi, temperatura, vlažnost, tlak, preračunana višina iz barometra ali GPS sprejemnika, naklon po 2 smereh, smer kompasa, količino moči pogonskega motorja v samodejnem načinu, hitrost iz GPS sprejemnika, prikaz prenosa podatkov;
- izbira med ročnim, stabiliziranim načinom in avtopilotom;
- uporaba funkcije za popravljanje smeri, višinskega naklona in nadmorske višine preko zaslonskega vmesnika in z glasovnimi ukazi;
- fotografiranje in snemanje v polni ločljivosti;
- nastavitve parametrov letenja:
    - meje, občutljivost, smer servo motorjev in meje plina;
    - meje branja senzorjev mobilne naprave.
- zaustavitev RPi;
- prikaz zemljevida in lokacije RPi na njem (slika 24);

- postavitev točke ali več točk avtonomni let ter prikaz poti;
- preračun in prikaz razdalje od zadnje točke do doma in do vzletne točke in poti letenja;
- pridobitev nadmorske višine terena na določeni lokaciji iz Google Maps API.



Slika 23: Uporabniški vmesnik končne Android aplikacije



Slika 24: Uporabniški vmesnik za avtonomni let

3.5 Programska rešitev za komunikacijo preko omrežja

Del programske rešitve, ki ga nismo predvideli v idejni zasnovi je bil komunikacijski del. Ker nismo pričakovali, da se bo razvoj tako zapletel v tem delu, se je časovni razpon razvoja precej podaljšal, povečala pa se je tudi zahtevnost sistema. Opisali bomo, kako je potekalo načrtovanje, podrobneje opisali raziskovanje in razvoj ter težave in rešitve, ki so se pri tem pojavile. V tem delu bo največ zahtevnosti, saj smo sproti odkrivali, kaj je/ni ustrezno, kaj potrebujemo in česa ne potrebujemo – temu primerno je tudi število podpoglavij, da bo predstavitev dela bolj pregledna in urejena. Na koncu sledi predstavitev končne arhitekture tega dela.

3.5.1 Analiza in načrtovanje

Osnovni cilj oziroma zahteva je bila, da rešitev deluje v različnih omrežjih, torej ne glede na lokacijo, tip in internetnega ponudnika. Delovati morajo vse kombinacije lokalno, javno: Wi-Fi - Wi-Fi, mobilno - Wi-Fi, mobilno - mobilno. Želimo neomejen doseg, saj s tem ustrezamo zahtevam, ki so bile postavljene tako zaradi želenih specifikacij, kot tudi zaradi precej možnih želja uporabnikov in konkurenčne prednosti.

Zaradi nefunkcionalne zahteve (nizkega zamika oz. latence) je želen tip omrežja 4G - LTE, ki omogoča pod 20ms latence. Sama pasovna širina 150/50 Mb/s ni ključna, saj promet dosega le delček sposobnosti, poleg tega pa je pomembna tudi stroškovna učinkovitost.

3.5.2 Raziskovanje in razvoj

3.5.2.1 Uvod in prvi poskusi

Razvoj je potekal iterativno oziroma prototipno po modulih. Potrebno je bilo raziskati način možne povezave preko različnih omrežij. Prva ideja so bili klasični vtičniki, ki omogočajo komunikacijo na omrežni plasti.

Spisan je bil testni program, vendar se je izkazalo, da to ne bo možna rešitev. Za medomrežno povezovanje namreč potrebujemo posredovanje vrat (angl. port-forwarding) na usmerjevalniku, dostop do tega pa ima v mobilnih omrežjih samo operater. Odkrita storitev No-IP bi nam omogočala dinamičen dostop do strežnika, ki bi se izvajal na RPi, vendar pa to ni bilo iskanje rešitve v pravi smeri, ampak zgolj širjenje obzorja. Na informacijo o odprtosti vrat (angl. port) je internetni mobilni ponudnih odgovoril, da taka povezava ni možna, saj je zaradi varnosti vzpostavljeno zasebno lokalno omrežje znotraj mobilnega operaterja in so povezave navzven nemogoče. Tukaj se je podrla prvotna zahteva zaradi omejenosti. Sledilo je raziskovanje vzroka, problema in kako se ga reši. Raziskovanje je privedlo do sprva navideznih rešitev, ki pa so se izkazale za nepravilne - socks5 [70] in proxy [71] namestniški posredniški strežnik ter VPN (navidezno zasebno omrežje), ki omogočajo samo polovično

rešitev problema - torej na eni strani, ali pa zahtevajo dodatno zahtevano konfiguracijo od uporabnika (VPN), zato se je raziskovanje nadaljevalo.

3.5.2.2 NAT naprave

Vse je vodilo do odprtosti vrat, požarnih zidov in naprav NAT (angl. Network Address Translation), ki preslikujejo množico zasebnih naslovov IP na vrata, da lahko z enih javnim naslovom IP več naprav deluje nemoteno, ko se povezujejo na internet. NAT je hkrati tudi varnostni element, poznamo pa štiri vrste teh naprav:

- statični, ki dovoljuje ves promet v obe smeri (notri, ven);
- dinamični, ki filtrira naslove IP (notranja naprava sprejme pakete iz naslova, na katere je predhodno poslala paket);
- dinamični, ki filtrira naslove IP in vrata (notranja naprava sprejme pakete iz naslova in vrat, na katere je predhodno poslala paket);
- dinamični simetrični, ki so zahtevni za naš problem, saj uporabljajo dvosmerno preslikavo (angl. mapping) in jih uporabljajo velika podjetja in korporacije;

Naslednji logični način bi bil opustiti direktno povezavo, vzpostaviti strežnik, ki bi bil posrednik za vse odjemalce, ki bi se nanj povezali. Izkazalo se je, da bi to bila draga rešitev, saj bi ves promet potekal čez strežnik. Vse poti so vodile k novi vrsti komunikacije – P2P (angl. peer-to-peer), to je direktnega načina komunikacije s posebnimi triki (podobno, kot delujejo storitve Skype z dodatkom super vozlišč (angl. supernode).

3.5.2.3 P2P in WebRTC

Komunikacija v realnem času (angl. Real Time Communication - RTC) je bila dolgo izziv razvijalcev [72]. Zahtevala je uporabo dragih avdio in video licenc. Google je tehnike RTC uporabljal že v storitvi Google Talk, leta 2011 pa je z odprtjem tehnologij, ki jo je razvilo podjetje GIPS, postavil temelje za prihodnost. V maju istega leta pa je Ericsson razvil prvi implementacijo WebRTC-ja. RTC uporablja tudi Skype in Facebook. Potreba po odprtokodni rešitvi P2P komunikacije za potrebe spleta so vodile v razvoj protokola WebRTC(aplikacijskega vmesnika), tako za brskalnike, mobilne naprave in IoT (angl. Internet of Things) naprave. Iniciativo WebRTC podpirajo Google, Mozilla, Opera in ostala podjetja.

WebRTC podpira večina brskalnikov, vendar je delovanje različno od verzije do verzije (Chrome, Firefox, Opera, Android, iOS).

Protokol omogoča bogato komunikacijo v realnem času (od podatkovne, glasovne do video komunikacije, brez dodatnih vtičnikov, prenosov in namestitev), saj brskalniki implementirajo WebRTC v svojem jedru [73].Veliko prednost je uporaba Googlovega video kodeka VP8, ki

omogoča nizko latenco komunikacije zaradi močnega algoritma za kodiranje/dekodiranje in stiskanje. Za razliko od kodeka H.264 in MJPEG je brezplačen za uporabo.

3.5.2.4 Signalizacijski strežnik

Za uspešno komunikacijo s protokolom WebRTC potrebujemo signalizacijski strežnik. Ta poskrbi, da se izmenjajo informacije za nadzor povezave, informacije o IP naslovih in vratih ter nivo podpore za avdio in video komunikacijo.

3.5.2.5 STUN strežnik

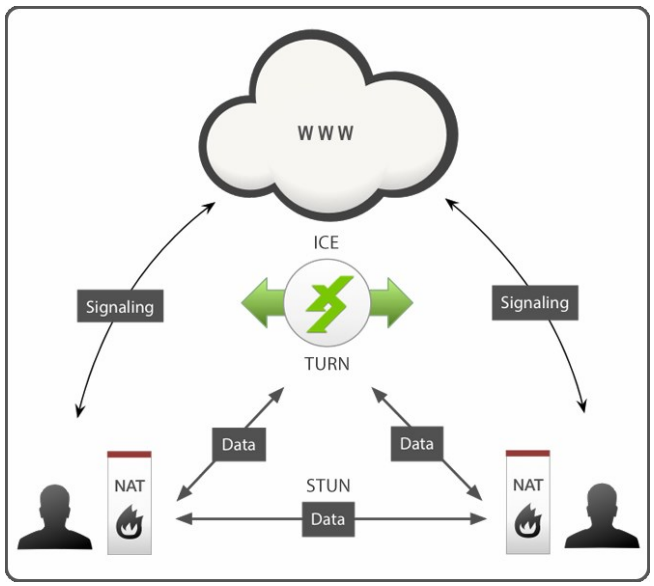
Prvi način komunikacije je preko protokola UDP - ta je nizkolatenčni, z majhnimi omrežnimi okvirji, nezanesljiv in brez potrjevalnih algoritmov na samem transportnem protokolu. Prav zato lahko z njegovo uporabo dosežemo visoke hitrosti. Problem nastane, ko med komunikacijo med končnima napravama obstaja NAT. STUN (Simple Traversal of UDP through NATs) [74] je standardizirana množica tehnik v omrežnem protokolu, ki omogoča končnim napravam odkritje svojega javnega naslova IP in vrat, ki mu ga je dodelila naprava NAT za javni internetni dostop. To STUN strežnik naredi tako, da odjemalcu, ki mu je poslal zahtevo, v odgovoru pošlje podatke o odjemalcih (naslov IP, vrata), kot jih vidi sam. Zaradi tega, ker samo izmenjuje podatke, lahko podpira na tisoče zahtev hkrati in celo na manj zmogljivih strežnikih. Še vedno pa STUN strežnik ni vedno rešitev. Deluje namreč v primeru treh vrst NAT-a, ne deluje pa na simetričnih NAT-ih. Težavne so povezave, pri katerih sta obe končni napravi za simetrično NAT napravo. Teh povezav je po statističnih podatkih do 16% [151]. Lahko bi trdili, da obstaja možnost, da kateri od mobilnih internetnih ponudnikov mobilnih storitev uporabljajo prav simetrične NAT naprave v svoji topologiji. Za to potrebujemo drugačen strežnik.

3.5.2.6 TURN strežnik

STUN strežnik ni dovolj, če potrebujemo delovanje v 100% primerov ne glede na topologijo omrežja in vključenost NAT naprav in požarnih zidov. Potrebujemo TURN (Traversal Using Relays around NAT) strežnik [75], ki deluje tako preko protokola UDP in TCP, gre pa za čisto drugačen pristop. Namenjen je povezavi računalnikov tako, da deluje kot posrednik (angl. relay). Posredništvo pa deluje na naslednji način. Odjemalec A pošlje določeno zahtevo na TURN strežnik, ta pa mu v primeru prostora v smislu zmogljivosti odgovori s podatki o transportnem naslovu, preko katerega bo komuniciral z odjemalcem B. TURN strežnik se nato poveže še z odjemalcem B in obvesti odjemalca A, da je povezava vzpostavljena. Vsak odjemalec, ki je povezan na TURN strežnik ima 2 načina, na katera lahko pošilja podatke. Sporočila se nato pošiljajo preko UDP okvirov. V kratkem opisu smo videli, kako je TURN strežnik robustnejši od STUN strežnika, vendar pa ima tudi svoje slabosti. Način komunikacije, ki v celoti poteka preko posrednika, zahteva precej večjo pasovno širino kot protokol STUN, ki odjemalcem samo izmenjuje podatke o javnih IP naslovih in vratih, da se lahko povežejo med sabo. Obstaja pa način, ki rešuje slabosti STUN in TURN strežnika. To pa je ICE.

3.5.2.7 ICE protokol za STUN/TURN strežnike

Protokol in tehnika povezovanja v komunikacijskih omrežjih na področju NAT razreševanja se imenuje ICE (Interactive Connectivity Establishment) [76]. Uporablja protokola STUN in TURN tako, da najprej poskuša vzpostaviti povezavo preko protokola STUN, če pa zaradi kakršnegakoli razloga to ni mogoče, uporabi protokol TURN. To prikazuje slika 25. Tako omogoča povezljivost v 100% primerov, hkrati pa ne obremenjuje TURN strežnika vedno, temveč samo za povezave, za katere ni drugega načina. Lahko se uporablja za katerikoli protokol, ki uporablja model ponudba/odgovor (angl. offer/answer).



Slika 25: Prikaz vzpostavitve komunikacije med dvema spletnima odjemalcema

3.5.2.8 Podporne knjižnice za WebRTC

Uporaba protokola WebRTC ni zelo zahtevna, vendar tudi ne lahka. Ker cilj te diplomske naloge ni odkrivanje že odkritega, si pomagamo z obstoječimi rešitvami, da rešitev izdelamo bolj učinkovito, hitreje, enostavneje in modularno. Če bi razvijali izključno WebRTC aplikacijo, potem bi bila uporaba surovega WebRTC aplikacijskega vmesnika logična izbira. Tako pa je WebRTC le del komunikacijskega dela naše rešitve in iščemo ustrezne podporne rešitve, ki so vzdrževane in aktualne. Raziskovanje je privedlo do več obstoječih rešitev na področju WebRTC-ja,

Nekatere od teh so bile socket.io [77], PeerJS [78], skylink.js [79], easyRTC [80], SimpleWebRTC [81], rtceverywhere [82] itd. Izbrali smo rešitev PeerJS. Na to so vplivale enostavnost vmesnika, dobra dokumentacija in kakšen zgled. Kasneje se je izkazalo, da nekaj stvari ni dokumentiranih in komunikacija brez drugačne konfiguracije vrat, uporabe protokola SSL, vzpostavitve svojega strežnika za namen signalizacije itd. deluje samo v lokalnih omrežjih. Dejansko bi bila kakšna druga rešitev lažja za implementacijo.

Primeri aplikacij/storitev zgrajenih na WebRTC, ki smo jih našli so: OpenTokRTC [83], Talky.io [84], AppRTC [85], Uber [86], in Twilio [87].

3.5.2.9 Razvoj in vzpostavitev osnovne arhitekture

Vzpostavitev komunikacije med dvema brskalnikoma je potekala modularno in ločeno od ostalega projekta. Da ne bi pisali nepotrebne kode, je programiranje potekalo s pomočjo dokumentacije. Tako je nastal dokaj delujoč program.

Za enostavno implementacijo poskusimo uporabiti signalizacijski strežnik, ki ga PeerJS daje zastonj v uporabo. Za ta namen pridobimo API ključ, s katerim se pridobimo dostop pošiljanja podatkov na njihov signalizacijski strežnik. Za delovanje smo zaradi enostavnosti uporabili enega ali več javnih zastonjskih Googlovih STUN strežniških storitev. Ko povezava deluje in smo prepričani, da je PeerJS res možna rešitev, moramo vzpostavimo svoj signalizacijski strežnik. Njihov javni namreč sprejema povezave na vratih 3000, kar pa onemogoča povezavo, če usmerjevalnik, preko katerega dostopamo do interneta, blokira vrata 3000 in nimamo dostopa do posredovanja vrat (angl. port forwardinga). Prav tako to predstavlja dodatno konfiguracijo omrežja, kar je za nas neustrezna rešitev. Lastni signalizacijski strežnik, ki teče v `Node.js` je bil postavljen na Azure cloud virtualnem Linux strežniku. Tam je bilo potrebno odpreti določena vrata za vhodni/izhodni promet tako v nastavitvah oblaka s pomočjo grafičnega vmesnika, kot na samem strežniku s pomočjo komandnega aplikacijskega programa iptables.

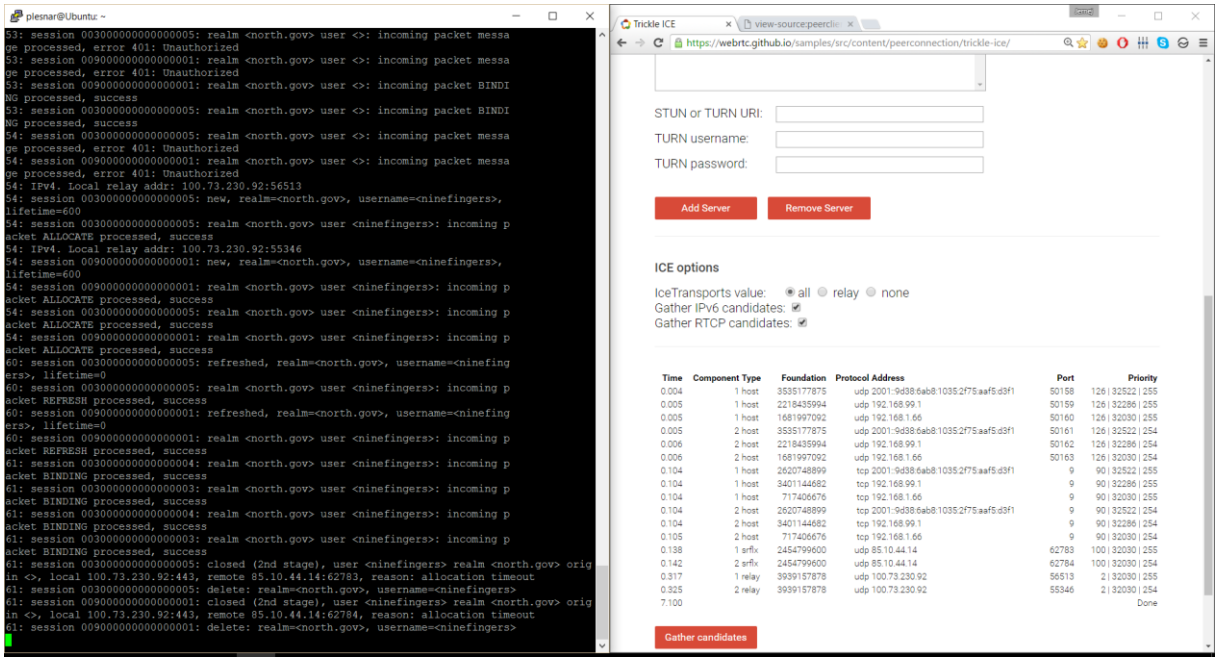
3.5.2.10 Vzpostavitev STUN/TURN strežnika

Povezava je delovala v lokalnem omrežju, kar je bilo dovolj za nadaljnji razvoj na ostalih področjih. Pri testiranju povezav med različnimi omrežji smo ugotovili, da bo potrebno dodati tudi TURN strežnik. Ta storitev se za razliko od STUN strežniške storitev ne ponuja zastonj, ker celotna komunikacija poteka preko TURN posrednika in za ponudnika teh storitev nastanejo stroški. Nekateri ponudniki teh storitev so: XirSys [94], ZeroTier [95], Twilio [87], AnyFirewall [96], Estos [97] in Icelink [98].

Obstajajo tudi odprtokodne rešitve STUN/TURN/ICE, da jih lahko postavimo na lastni oz. gostujoči privatni/virtualni strežnik: turnserver [88], pysip [89], restund [90], creytiv [91], rfc5766-turn-server [92] in coturn [93].

Za naše potrebe smo izbrali coturn implementacijo TURN strežnika, ki se je zdela robustna (podpira delilnika bremena (angl. load-balancing), različne operacijske sisteme in še mnogo več), je vzdrževana in za uporabo enostavna rešitev. Na odločitev je vplivalo tudi to, da ga je razvil in ga tudi vzdržuje Google. Postavili smo jo na Azurjev Linux strežnik, kjer je že uspešno tekel signalizacijski strežnik. Kljub enostavni namestitvi, delovanje ni bilo trivialno vzpostaviti. STUN strežnik je deloval normalno, TURN strežnik pa samo lokalno. To smo ugotovili s testi. Zaradi pomanjkanja časa v tem obdobju razvoja se je odprava napake vlekla čez več mesecev v dolgih presledkih. S pomočjo spletnega testnega orodja za testiranje STUN/TURN funkcionalnosti imenovane Trickle-ice [99] smo testirali različne konfiguracije

(slika 26). Končno smo odkrili, da je bil vzrok za nedelovanje strežnika to, da WebRTC več ne podpira TURN funkcionalnosti preko nešifriranih povezav. Tako je bilo konfiguracijo strežnika spremeniti, da je uporabljal SSL protokol. Privzeta vrata za protokol TURN so 3478 - spet ovira za usmerjevalnik na mobilnem omrežju. Tudi tukaj smo porabili kar nekaj časa za ugotovitev in odpravo napake. Za rešitev smo vzpostavili nov Linux strežnik v Azure oblaku, namestili TURN strežnik in ga konfigurirali na vrata 443 (SSL). Pozneje smo odkrili zastonjsko storitev Numb [100] Kalifornijskega podjetja Viagenie, ki deluje odlično. Zaradi zanesljivosti ostajamo na svojem strežniku.

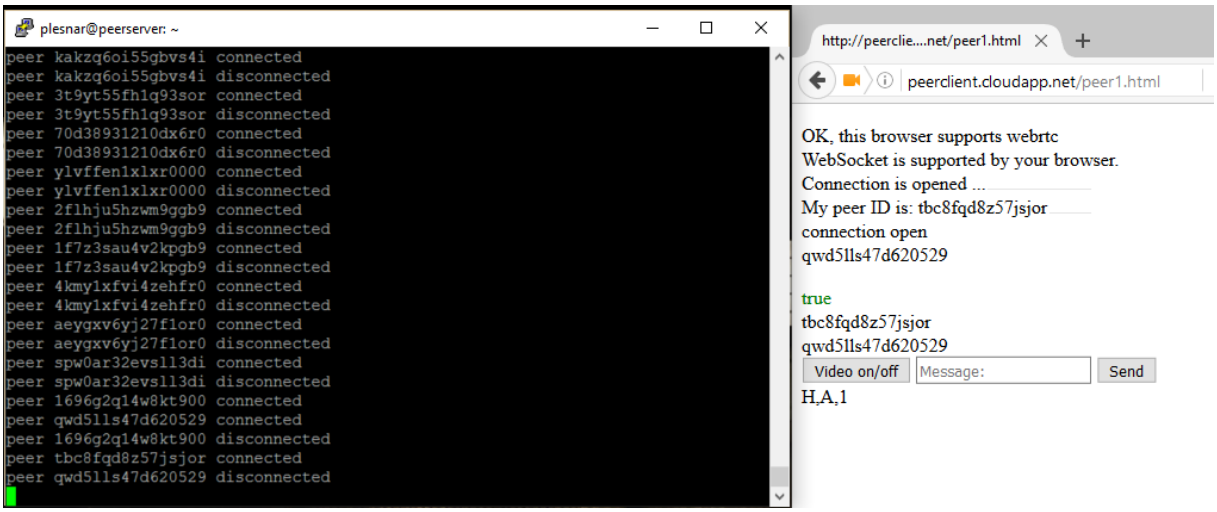


Slika 26: Prikaz testiranja delovanja STUN/TURN strežnika

3.5.2.11 Vzpostavitev signalizacijskega strežnika in odjemalcev

Vsak WebRTC odjemalec uporablja za uporabljen signalizacijski strežnik unikaten identifikacijski niz. Tako se lahko dva odjemalca povežeta med sabo, če vsaj eden ve ID od drugega. Za izmenjevanje te informacije seveda ne more poskrbeti WebRTC, temveč potrebujemo lastno komunikacijsko rešitev, odvisno od tematike in vrste našega problema. Po tem naj bi komunikacija delovala brezhibno. Preko protokola WebRTC lahko pošiljamo nekaj različnih vrst podatkovnih struktur. Stvari pa se zapletejo, če se hočemo večkrat povezati na signalizacijski strežnik z istim ID-jem. To v splošnem ni priporočeno, saj signalizacijski ID ni namenjen identifikaciji samih naprav, ampak vzpostavitvi povezave. Vseeno pa to želimo, če naša rešitev vključuje povezovanje na isto napravo in nočemo vedno znova izmenjevati ID-jev. To bi bilo za isto napravo nesmiselno in nepotrebno delo, prav tako pa netransparentno in neugodno za končnega uporabnika. Problem nastane, ker odjemalec svojega ID-ja ne odjavi iz signalizacijskega strežnika in za drugega odjemalca vse izgleda, kot da je povezava še vedno

vzpostavljena (vsaj v primeru knjižnice PeerJS) (slika 27). Ker posebej ne piše, da odjava ne steče transparentno, smo ta problem v intervalih opravljali do poznih faz razvoja. Na to je delno vplivala modularna zgradba, ki je omogočala nadaljnji razvoj kljub tej napaki. Le večkrat je bilo potrebno ponovno zagnati signalizacijski strežnik (ne fizičnega strežnika, ampak aplikacijsko programsko opremo). To s primerno skripto ne traja več od sekunde.



Slika 27: Testiranje (ne)pravilnega delovanja signalizacijskega strežnika in odjemalcev

V poznih fazah smo ugotovili, da je potrebno na dogodku, ko se ustavlja Javascript v brskalniku, implicitno klicati odjavo iz signalizacijskega strežnika in tako omogočiti ponovno prijavo z istim ID-jem. To seveda deluje brezhibno na računalnikih brskalnikih, na mobilnih napravah in v ostalih vmesnikih pa je drugačna zgodba [101]. Za vzdrževanje povezave je potrebno osveževanje strani, saj po daljši neaktivnosti postane nedostopna. Tako se je že v začetnih fazah razvilo dve različni spletni aplikaciji, ker gre za dve različni funkcionalnosti, kar se je tudi pozneje se je to izkazalo za dobro izbiro. Zaradi modularnosti in vzdrževanja se je postavilo tretji strežnik v Azure oblaku, ki poganja spletni strežnik Apache in gosti ti dve spletni aplikaciji.

3.5.2.12 Pretok videa med odjemalcema

WebRTC podpira tudi video klice. V našem primeru potrebujemo enosmerni video klic. To pomeni, da če hočemo iz naprave A tok videa prenašati na napravo B in ga tam predvajati (v realnem času), mora klic sprožiti naprava A, naprava B pa mora na klic odgovoriti. Pri implementaciji smo ugotovili nedelovanje pretoka med nekaterimi brskalniki. Za pretok videa mora namreč brskalnik pridobiti dovoljenje in vse potrebno preko gonilnika video kamere in potem tok videa pošiljati preko WebRTC aplikacijskega vmesnika. Zajem slike ni uspel zaradi pomanjkanja podpore različnih verzij aplikacijskega vmesnika za video, in verzije brskalnika. Paziti moramo tudi na združljivost med različnimi brskalniki [102]. Tudi na splošno je bila večja zanesljivost in kakovost povezave na takšen način, da odjemalec, ki pošilja pretočni video, teče v brskalniku Mozilla Firefox (kar ni bila dokončna odločitev), odjemalec, ki pretočni video sprejema, pa v brskalniku Google Chrome. V ozadju seveda ne

gre za brskalnik, ampak pogon (in pod njim video podporo), ki ga brskalnik uporablja, torej Gecko za Firefox in WebKit za Chrome. Žal video pretok še vedno ni deloval. Za analizo in odpravo problemov je bilo potrebno preveriti log datoteke v WebRTC vmesniku v brskalniku Chrome na naslovu `chrome://webrtc-logs`, ker je bila napaka očitno v knjižnici PeerJS.

Po mesecih intervalnega ugotavljanja in testiranja smo v poznejših fazah odkrili, da obstaja novejša knjižnica z izboljšavami (in najbrž vsaj delno odpravljenimi napakami), imenuje se Skyway PeerJS [103]. Gre za produkt kitajskega komunikacijskega podjetja Nttcom, ki jo je razvilo za potrebe spletnega servisa SkyWay. Vendar pa ima tudi ta svoje pomanjkljivosti za naš primer: prva je, da njihov signalizacijski strežnik vsaj v tisti fazi razvoja ni deloval, drugo pa je, da identifikacijske številke ID ni možno ročno izbrati, ampak je vsakih dodeljena samodejno. Namen in cilj tega je seveda dober (preprečiti možnost neuspeha, če bi hotela dva odjemalca isti ID), za naš projekt pa je to neugodno. Vseeno pa smo se odločili za delno uporabo te knjižnice, ker uspešno rešuje problem pretoka videa. Delno uporabo pravimo zato, ker smo za odpravo obeh pomanjkljivosti na eni strani morali uporabiti knjižnico PeerJS, na drugi pa Skyway PeerjJS.

V poznih fazah razvoja se je dodalo še povezave z vtičniki. Razvoj komunikacijskega dela na nivoju brskalnika je bil tako v veliki meri neodvisen in zato delno vzporeden od ostalega projekta. Opisana komunikacijska rešitev omogoča razširljivost in jo lahko uporabimo za razvoj kot kombinacijo na drugih mobilnih napravah s sistemi iOS, Windows Phone in mikrokrmilnikih ranga RPi (Beaglebone, BananaPI [104], Arduino Yun [105], Intel Galileo), ki imajo zmožnost poganjati WebRTC.

3.5.2.13 Razvoj osnovne komunikacije med odjemalcem in mobilno aplikacijo

Še vedno pa rešitev omogoča komunikacijo samo med brskalniki, za naše potrebe pa potrebujemo komunikacijo med Android napravo in RPi Python programom. Ideja je bila naslednja: na strani Android naprave smo ugotovili, da bo najbolj modularno in zanesljivo, če bomo uporabili WebView element, ki je Androidov grafični element in omogoča podporo Googlovega pogona WebKit. Sicer obstajajo tudi programska orodja kot je Crosswalk [106] in Skywayev SDK, ampak Xamarin Android že privzeto podpira WebView.

Nekaj časa je trajalo učenje potrebnega za komunikacijo med Javascriptom WebView elementa in preostalim programom. Potrebno je namreč spremeniti nastavitve WebView elementa in med drugim v njem omogočiti Javascript. Potrebno je bila implementacija javascript vmesnika in njegovih metod, da jih WebView deklarira v JS in jih lahko potem kličemo naravnost iz JS. Tudi vmesnik mora biti modularen in nespecifičen za naš problem. Po tem komunikacija z WebViewjem ni bila več težavna, razen tega, da se funkcija za odjavo iz signalizacijskega strežnika iz komunikacijskega dela rešitve ni izvajala. Tako je bilo posebej poskrbeti za reševanje tega problema, ki nastane zaradi drugačnega obnašanja WebView elementa od samega brskalnika, pa tudi zaradi življenjskega cikla Android

aplikacij. Reševanje problema je namreč trenutno oviralo to, da se ob osvežitvi spletne strani podere delovanje zaradi akcije, ki ni bila sprožena v glavni niti. To se je popravljalo v poznih fazah razvoja na drugačen način. Samo razhroščevanje spletnega odjemalca na Android napravi je potekalo s pomočjo izpisa v spletni aplikaciji, prav tako pa smo si pomagali z Googlovim oddaljenim razhroščevalnim orodjem za brskalnike na osnovi WebKit pogona.

3.5.2.14 Razvoj osnovne komunikacije med odjemalcem in RPi

Več problemov je bilo na strani komunikacijskega kanala in sicer na RPi, saj je bilo potrebno več ciklov raziskovanja, razvoja in testiranja, da smo ugotovili ustrezen način, ki bo sploh deloval. Ideja je bila še vedno uporabiti obstoječo rešitev. Operacijski sistem bi torej poganjal brskalnik, Python pa bi na nek način komuniciral z njim in obratno. Samo za odkrivanje pravega brskalnika je bilo veliko dela. Testirani so bili različni brskalniki: Dillo [107], Epiphany Web Browser [108], NetSurfWebBrowser [109], Chromium Web Browser [110] in Iceweasel [111]. Iceweasel je prejšnji Firefox za Debian sisteme. Ker se je za nekaj časa med razvojem diplomske naloge ustavil razvoj Firefoxa, sedaj se spet imenuje Firefox (Firefox-ESR), vendar obstaja še tudi verzija Iceweasel-Firefox-ESR. Nekaj časa smo se iskali tudi pri tem, ker nikjer uradno ni bilo obvestila, da trenutno Firefox ni več na voljo. Iceweasel in Firefox sta izmed vseh testiranih edina, ki podpirata protokol WebRTC na Debian sistemu. Sledilo je raziskovanje in testiranje delujoče Python knjižnice za komunikacijo z Javascriptom. Preizkušene so bile naslednje: PyQt4 [112] (potrebna je draga licenca za Qt), PySide [113], PyExecJS [114] in PyJS [115] (ni podpore za WebRTC).

Kazalo je slabo, zato je sledila ideja, da bi osnova tekla v brskalniku in Python v njemu. Za ta namen smo testirali knjižnici Skulpt [132] in Brython [133]. Problem obeh je bila počasnost izvajanja proti »pravemu« Pythonu (CPython), zato je bila ideja v celoti ovržena. Odkrita je bila nova možnost, to je bila precej popularna in robustna rešitev za testiranje programske opreme, natančneje spletnih aplikacij. Imenuje se Selenium [116].

Ta ima vmesnik tudi za Python, zato se je naš razvoj lahko nadaljeval. Ob prvih nekaj primerih se je izkazala za dovolj hitro komunikacijsko rešitev med Pythonom in Javascriptom, kasneje pa smo jo še pohitrili tako, da smo počasnejše funkcije `SendKeys` zamenjali naravnost z `javascriptExecutor` objekti, ter uporabili brezglavi način (angl. headless) (na začetku v `Xvfb`, pozneje z namenskim ukazom Seleniuma) ter poskus uporabe spletnega pogona PhantomJS, ki pa žal ne podpira protokola WebRTC.

3.5.2.15 Prenos video pretoka iz PiCamere v brskalnik

Celoten razvoj je potekal modularno, zato se je tudi Selenium del, (razen osnovnega testa delovanja) testiral na računalniku in spletni kameri prenosnika. Poseben izziv komunikacije direktno iz RPi je bil prenos slike, kar se je popravljalo v poznih fazah razvoja. Za ta namen so bile testirane in uporabljene naslednje knjižnice in možnosti.

Gstreamer [117] je bila prva izbira še v idejnih fazah, ko še nismo imeli P2P komunikacije, saj ima dobre reference, je odprtokoden in robusten. Testirano ponuja nizko obremenitev

procesorja, ogromno možnosti stiskanja, kodiranja, izhodnih in vhodnih tokov in formatov (MJPEG, H.264 itd.). Po novem ponuja tudi SDK za Android in iOS. Zelo veliko lokalnih projektov poganja prav gstreamer. Kombinacija programa `raspivid` in `gstreamer` preko cevovoda ponuja video pretok z nizko latenco na drugo napravo v lokalnem omrežju.

Nekaj opcij video pretoka je opisano oziroma predstavljeno na Wikipediji [119], ampak nobena primerna za nas:. Ena od možnosti je tudi netcat [118], spet samo za določen naslov IP. Problem je namreč dostop video pretoka iz Javascripta, da ga lahko posredujemo naprej. Eno od upanj za prenos videa je bil Janus. To je WebRTC prehod za video klice in konference, vendar nam ga takrat ni uspelo usposobiti na način, da bi tok pridobili iz RPi kamere. Za potrebe lokalnega testiranja smo uporabili spletni strežnik Nginx. Sedaj bi bila to verjetno možna opcija, vendar s predelavo obstoječe WebRTC komunikacije, česar pa ne želimo po nepotrebnem. Hkrati bi morali uporabiti Jitsi meet in xmpp bridge (npr. Jabber), kar dodatno zaplete stvari in omeji uporabnika.

Druga od možnih upanj je bila rešitev Kurento [120] (multimedijski strežnik za pretok videa), vendar žal ne obstaja za ARM procesorje. Možen bi bil prevod programa na PC-ju (angl. cross-compiling) ali uporaba z v zadnjih fazah odkritim emulatorjem ExaGear [121] podjetja Eltechs.

Ena od prenovljenih rešitev je UV4L [122], v času raziskovanja je bila dostopna stara spletna stran s staro vsebino, s katero je bilo tudi veliko neuspešnega vpeljevanja v projekt.

Po raziskovanju ogromno možnih opcij (ki jih ne bomo opisovali) preko različnih protokolov aplikacijske (HTTP, HTTPS) ni transportne plati (TCP, UDP) in preteku več mesecev počasnega in demotivirajočega razvoja smo odkrili nekaj gonilnikov in rešitev, ki omogočajo brskalniku dostop do kamere.

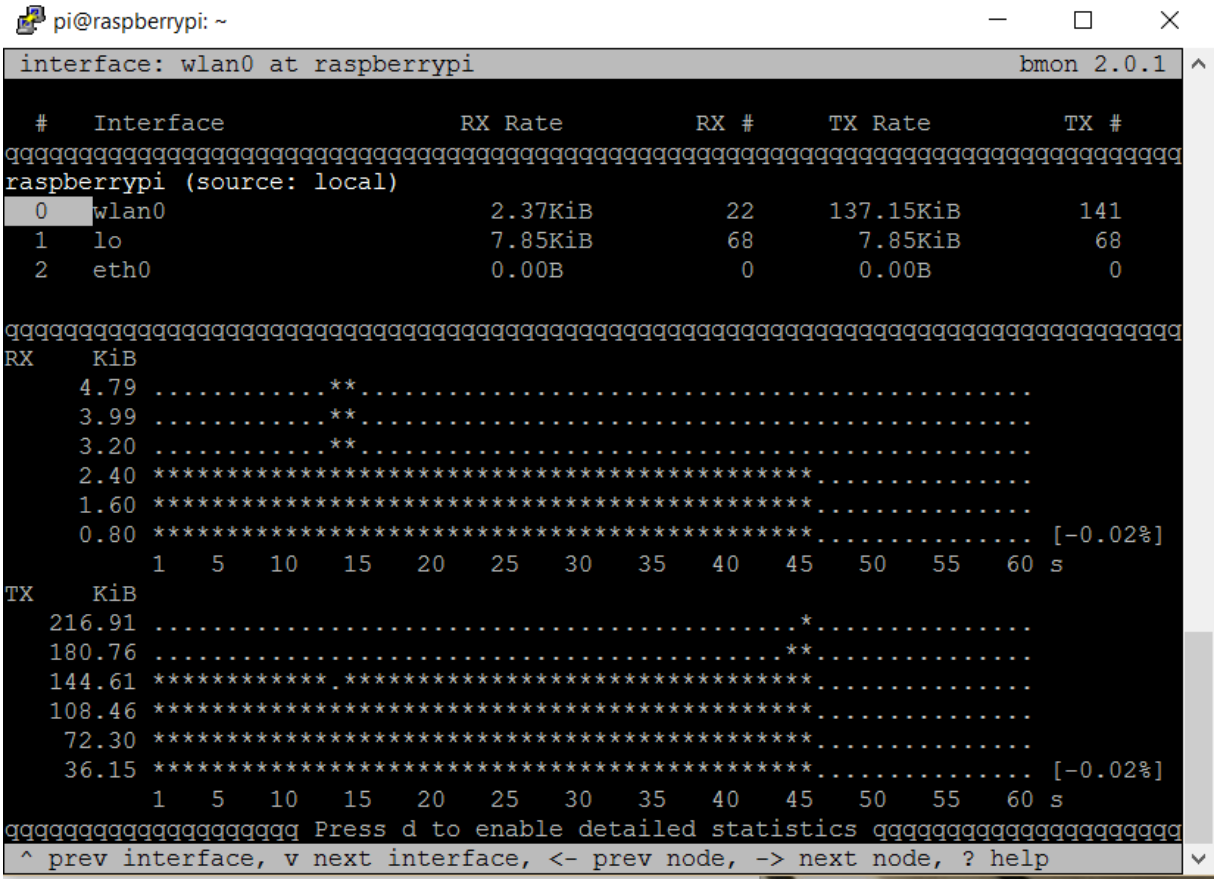
Možna rešitev je RPi Cam Web interface [123], ki omogoča pretok videa v brskalnik, vendar rešuje problem samo lokalno. Testirana in uporabljena je bila za kratek čas, vsebuje kar nekaj uporabnih funkcij, vendar za kakšen drugačen projekt.

Možna je tudi za nas prava rešitev z v4l2 [124] in podprtim gonilnikom, ki ga naložimo pri ob zagonu in omogočimo dostop do kamere. Na koncu zelo elegantna in enostavna in hkrati učinkovita rešitev za prejšnji problem, kjer so se nalagali nivo za nivojem. Pri video gonilnikih je dobro to, ker se lahko nastavlja različne parametre, da si prilagodimo rezultat svojim potrebam. Najbolj pomembni na nas so:

- ločljivost (angl. resolution), količina pretoka bitov (angl. bitrate) in količina slik na sekundo (angl. framerate);
- video format in stisljivost - kvaliteta videa, količina prometa, zahtevnost in obremenitev za procesor CPU ali GPU, če obstaja podpora za grafično pospeševanje, v našem primeru obstaja;

- h264 nivo (angl. level) - nivo kodiranja;
- intra - h264 uporablja tako imenovana tehnologija intra okvirjev (angl. keyframe). Tehnika deluje tako, da se polna slika pošilja samo na določen interval, vmes pa se pošiljajo samo razlike glede na zadnji intra okvir. To močno poveča učinkovitost prenosa in zmanjša promet in potrebno pasovno širino in tudi stroške. Žal poveča pa stiskanje in s tem obremenitev procesorja, ampak v primerjavi s prednostmi je to zanemarljivo, če nam to zmožnosti uporabljenega sistema to dovoljujejo.

Pri prenosu video pretoka smo sproti nadzirali količino prometa, kar prikazuje slika 28.

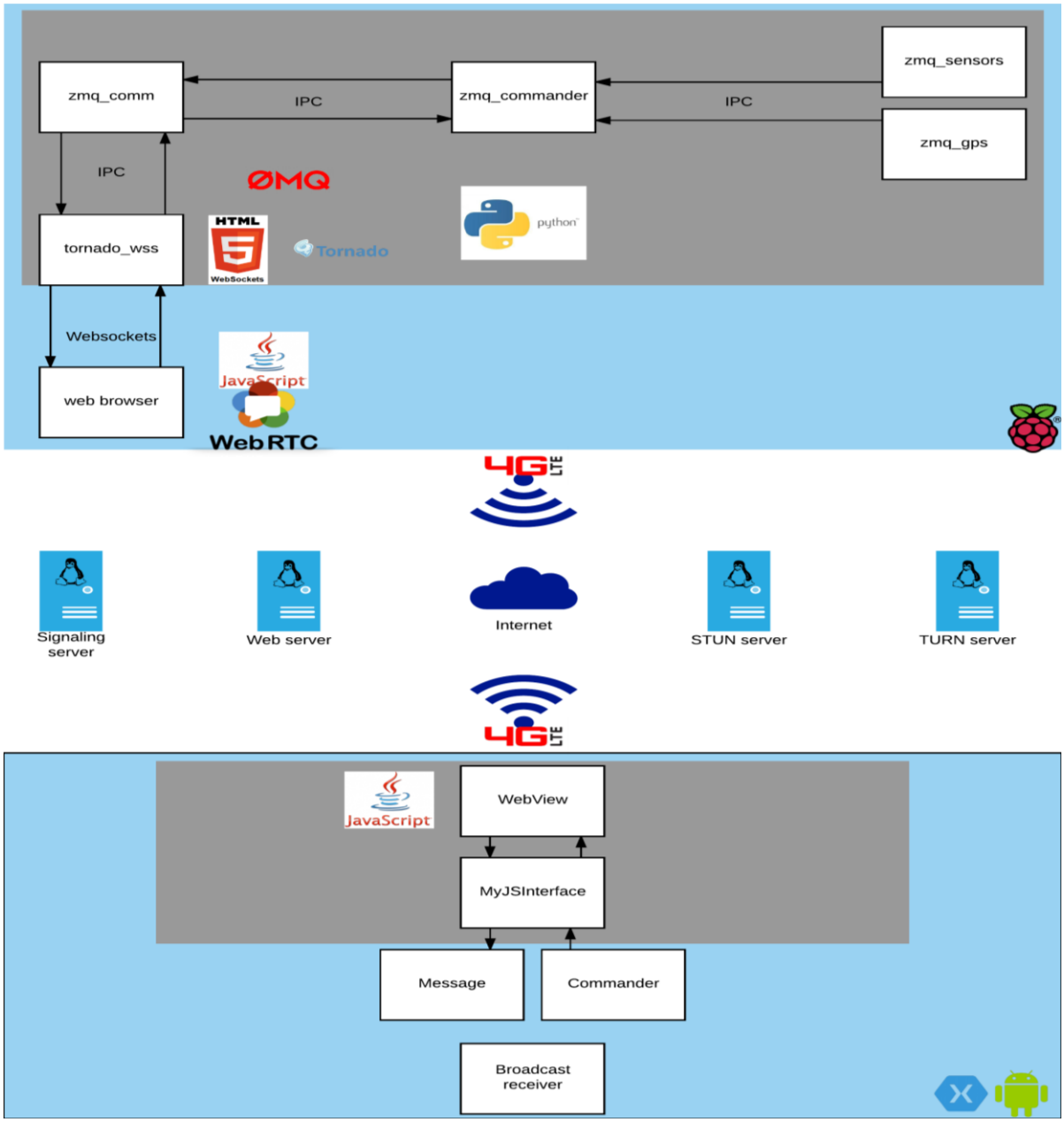


Slika 28: Nadzor prometa pri implementaciji video pretoka slike

Na podoben način deluje tudi VP8 enkoder tehnologije WebM [125], uporabljene v protokolu WebRTC. Težave s sliko so bile tako načeloma rešene, prišli pa smo do novih izzivov in z njimi do novih problemov. Selenium se je uporabljal do poznih faz razvoja. Po nadgradnji brskalnikov se je namreč izkazalo, da je podpora novih verzij za Selenium še zelo slaba, izboljšav pa ni pričakovati kmalu. Čas pošiljanja ukaza Python-Javascript ali obratno, je trajal prej 30ms, potem pa kar 70ms. To je nesprejemljivo in neuporabno na naš namen, saj je potrebno preverjati povezljivost, sprejemati ukaze itd. V poznih fazah se je komunikacija Python brskalnik zamenjala z lastno rešitvijo z uporabo spletnih vtičnikov.

3.5.3 Arhitektura

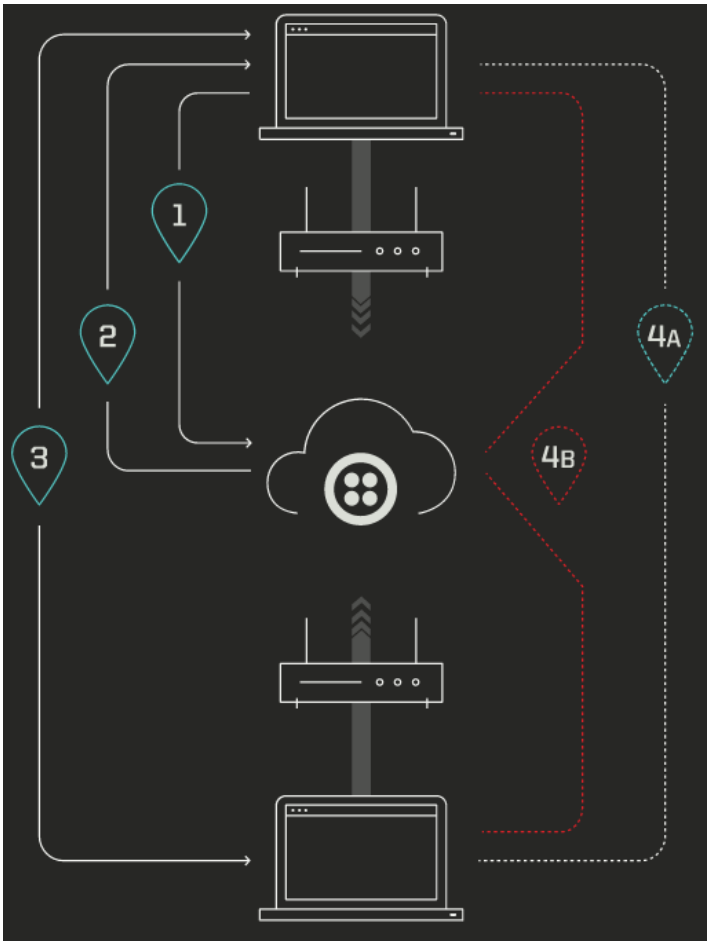
V kratkem opisu bomo pogledali komponente komunikacijskega dela, katerih namen in delovanje smo že opisali v prejšnjih delih besedila. Pri tem želimo opomniti, da gre za razvoj sistema, ki je nastal ob raziskovanju in najbrž obstajajo boljši načini za kakšno drugo rešitev, pa tudi izboljšave in novi pristopi za našo rešitev.



Slika 29: Arhitektura komunikacijskega dela rešitve na strani odjemalcev

Slika 29 prikazuje posplošeno arhitekturo komunikacijske rešitve, da se jasno vidijo različni načini komunikacije, uporabljenih tehnologij, programskih jezikov in sistemov. Komponente na RPi strani komunicirajo preko IPC komunikacije, medtem ko delujejo asinhrono in

neodvisno. Način povezave se razlikuje od sistema in sicer gre za TCP in povezave spletnih vtičnikov. Na Android strani vidimo komunikacijo med komponentami. Čeprav izgleda enostavnejša, gre v ozadju za bolj zapletene procese, ki se izvajajo v samem Android okolju. Spletni strežnik služi za hranjenje naše spletne aplikacije, ki komunicira z zunanjim svetom.



Slika 30: Vzpostavitev komunikacije med odjemalcema

Slika 30 prikazuje oblačni del komunikacije, ki je na prejšnji sliki samo delno prikazan z infrastrukturo sistema. Na tej sliki vidimo potek vzpostavitve povezave s pomočjo signalizacijskega, STUN in TURN strežnika.:

1. maprava pošlje zahtevek STUN strežniku ;
2. STUN strežnik vrne podatke;
3. test direktne povezave;
4. vzpostavitev povezave;

a. neposredno P2P;

b. preko (TURN) posrednika v primeru simetričnih NAT naprav.

Poglavje 4 Razvoj modela letala

Za namen testiranja, da se čim bolj pravilno in kvalitetno izdela nalogo glede na zmožnosti in dostopne vire, smo razvili namenski CAD model letala, ki bo služil namenu testiranja v zadnjih fazah razvoja celotne rešitve, pa tudi uporabi izdelka po zaključku izdelane rešitve.

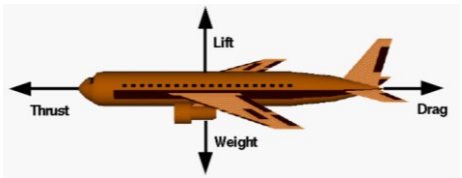
4.1 Razvoj CAD modela letečega krila

4.1.1 Osnove aeronavtike

Za namen razumevanja razvoja lastnega CAD modela letečega krila bomo zelo na kratko in jedrnato opisali osnove aeronavtike, ki so potrebne za boljše razumevanje teme. Celotna teorija (z vajami) osnov aeronavtike je na voljo na spletni platformi Edx predmeta fakultete Tu Delft z naslovom Introduction to Aeronautical Engineering [126]

4.1.1.1 Osnovne sile na letalo

Na letalo delujejo 4 sile (slika 31). Sila teže (letala), vzgona (večinoma glavnih kril), potiska (pogonskih motorjev) in upora (letala). Prvi dve in drugi dve sta med seboj nasprotni, zato pri uravnoteženju vseh sil letalo miruje (na tleh) ali se giblje enakomerno.



Slika 31: Sile na letalo

Vzgon

Sila vzgona je odvisna od koeficienta vzgona C_L, hitrosti zraka V in njegove gostote ρ in površine krila S. Koeficient vzgona nima dimenzij in je odvisen od profila krila in AOA (angl.

angle of attack) (vpadni kot, ki je kot med smerjo zračnega toka in tetiva profila krila). Med letom se spreminja gostota padajoče z večjo nadmorsko višino, ki je pri 0 m po ISA (mednarodnem atmosferičnem modelu) enaka 1.225 kg/m^3, hitrost zraka in površina krila, ki je odvisna od opcijskih visokovzgonskih naprav.

$$L = C_L \cdot \left(\frac{1}{2} \cdot \rho \cdot V^2\right) \cdot S$$

Upor

Sila upora je odvisna od koeficienta upora C_D (sestavljen iz trenja, tlačnega in induciranega upora), hitrosti zraka V^2, njegove gostote ρ in površine krila S. Ostalo je isto kot pri vzgonu.

$$D = C_D \cdot \left(\frac{1}{2} \cdot \rho \cdot V^2\right) \cdot S$$

Potisk in teža

Stremimo doseči čim manjšo težo, da za iste pogoje potrebujemo manj vzgona in posledično manj potiska, ker ta vpliva na vzgon posredno zaradi hitrosti. Tako porabimo manj energije in dosežemo daljši čas letenja in daljše razdalje.

4.1.1.2 Stabilnost letala

Poznamo dve vrsti vzdolžne stabilnosti (nagib gor - dol):

- statična – letalo je pozitivno statično stabilno, ko je nevtralna točka (točka uravnoteženih sil na krilo) za težiščem letala. Želimo pozitivno statično stabilnost, kar pomeni, da se letalo pri motnjah zaradi vetra ali kontrole samo stabilizira;
- dinamična – letalo je pozitivno statično stabilno, ko se dolgoročno poravna na stanje pred motnjo. To je za naše znanje in diplomsko delo prezahtevno in nerelevantno.

Poznamo tudi bočno stabilnost (nagib levo - desno), za katero hočemo, da je pozitivna, kar pomeni, kar se pri nagibu levo ali desno letalo samo stabilizira. Na to vpliva konus in lom krila zaradi porazdelitve vzgona in zračnega toka čez krilo.

4.1.1.3 Profili in krila

Bernoullijev zakon ohranitve energije za nestisljive fluide (tekočine in plini) pravi, da je vsota energije tlaka in hitrosti konstanta [127]. Krilo proizvaja vzgon tako, da se zračnim tokovom nad krilom poveča, pod krilom pa zmanjša hitrost. Po prej omenjenem zakonu se nad krilom

zmanjša, pod krilom pa poveča tlak, kar povzroči silo pravokotno zračnemu toku pod efektivnim kotom AOA.

Profil krila in AOA določata tok zraka po krilu in njegovo mejno plast. Mejna plast določa, kje se laminarni zračni tokovi spremenijo v turbulentne in se odcepijo od krila. To povzroči izgube vzgona in povečanje tlačnega upora ter generiranje zračnih vrtincev. Pri debelejših profilih in večjih AOA se na zgornji plasti krila tok odcepi od površine (kritično Reynoldsovo število) in ustvarja tlačni upor. Turbulentnih tokov načeloma nočemo, lahko pa je tudi zaželena lastnost. Turbulentna mejna plast ima večjo kinetično energijo blizu površine, saj ima tam tok večjo hitrost. Tako zakasnimo odcepitev mejne plasti in zmanjšamo tlačni upor. To se doseže s turbulatorji, to so grobe površine vzdolž dela krila, ki ustvarijo turbulenten tok.

Krilo je za razliko od profila 3 dimenzionalno in ima zato dodatni upor. Prvi razlog je upor zaradi vrtincev, ki nastajajo na koncih kril, drugi pa induciran upor, ki je odvisen od vpadnega kota in distribucije vzgona – geometrije krila. Stremimo k eliptični distribuciji vzgona, kar lahko dosežemo z obliko krila, zvijanjem ali konusom krila.

4.1.1.4 Mehanika leta

Kot zanimivost bomo opisali glavne karakteristike mehanike leta, ki bi jih lahko izračunali na modelu letala.

Najmanjša hitrost

$$V_{min} = \sqrt{\frac{W}{S} \cdot \frac{2}{\rho} \cdot \frac{1}{C_{L_{max}}}}$$

Najmanjšo hitrost, pri katerem lahko letalo leti pri horizontalnem letu, izračunamo tako, da v formulo vstavimo težo w, površino kril S, gostoto zraka ρ in največji možen koeficient vzgona krila C_L.

Največji doseg

$$V_{optimal} = \sqrt{\frac{W}{S} \cdot \frac{2}{\rho} \cdot \frac{1}{\left(\frac{C_L}{C_D}\right)_{max}}}$$

Največji doseg, pri katerem lahko letalo leti pri horizontalnem letu, izračunamo tako, da v formulo vstavimo težo w, površino kril S, gostoto zraka ρ in največji možen količnik koeficientov vzgona C_L in upora krila C_D.

Največji čas leta

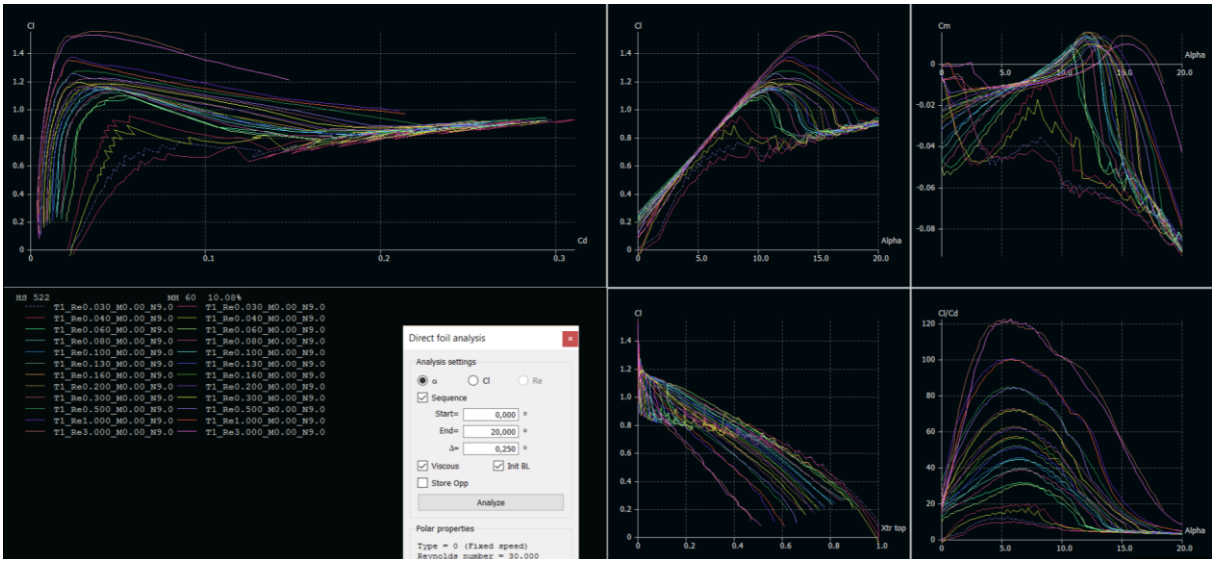
Za maksimalni čas leta želimo leteti pri minimalni hitrosti in minimalnem uporu.

4.1.2 Kriteriji

Radi bi enostavno rešitev, ki je učinkovita za naš namen (FPV letenje) in testirana s pomočjo orodij CAD orodja Solidworks: Flow simulation [128] in Parametric study [129]. Prvo orodje nam omogoča simulacijo fluidov (tekočin in plinov), s katerim preverimo naš model na realnih pogojih in dobimo rezultate tokov, sil in njihovih posledic na naš model. Z drugim orodjem lahko s pomočjo prvega orodja na podlagi več različic parametrov najdemo najboljšo variacijo testnega modela glede na podane cilje. Naš cilj je poskusiti oblikovati model in ga tudi izdelati.

4.1.3 Izbira in CFD analiza profila krila

Za profil krila smo izbrali popularen profil za leteča krila HS522 iz baze profilov [152] in ga analizirali v programu Xflyer (slika 32). Profili za leteča krila imajo posebno obliko, saj morajo kompenzirati navor (in to na kratki ročici), ki ga drugače ustvarja repno krilo (na dolgi ročici).



Slika 32: Analiza profila krila s pomočjo programa Xflyer

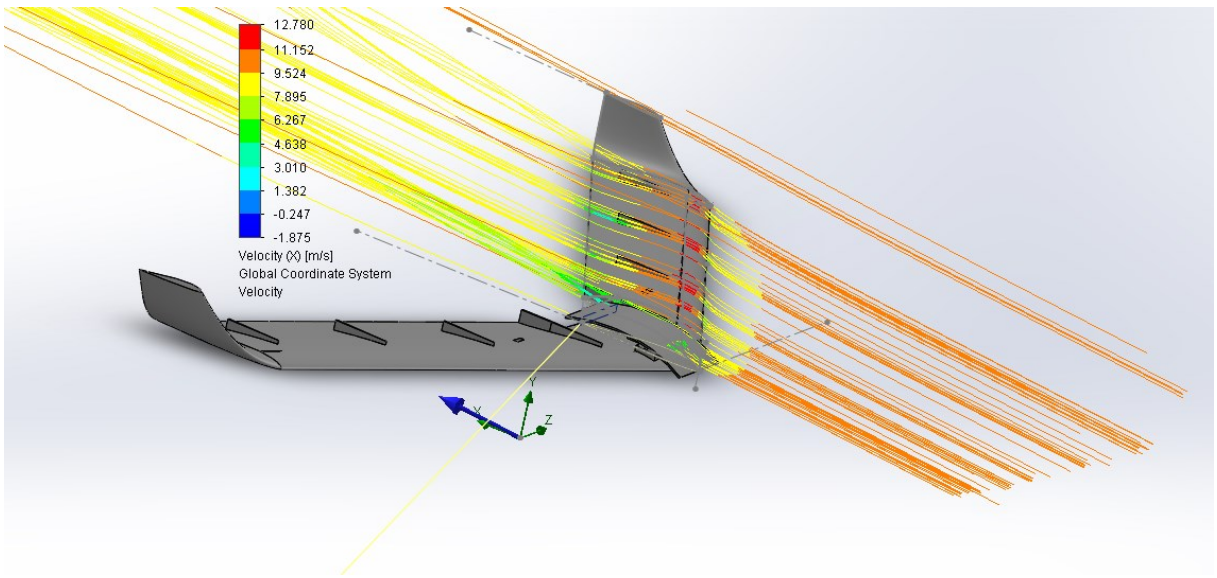
Iz analize na sliki 32 vidimo, da je največji količnik Cl/Cd (angl. glide ratio) pri alpha=5° (graf - desno spodaj). Prav tako vidimo, da se Cl veča samo do kota alpha=15° (graf - sredina zgoraj). To sta pomembna podatka za simulacijo in letenje, saj vemo teoretične vrednosti, torej lahko vsaj približno sklepamo, kdaj letalo leti učinkovito (za dolg čas leta ali let na dolge razdalje), če imamo podatke iz senzorja naklona.

4.1.4 Izbira letečega krila za namen FPV letenja

Leteče krilo je bilo za namen FPV letenja izbrano zaradi aerodinamične učinkovitosti in zato dobrih lastnosti za lete dolgih razdalj in časovne vzdržljivosti (angl. endurance). Kljub vsemu je težje za izdelavo, vodenje in nadzor zaradi ozkega profila, pomanjkanja stabilnosti in lege težišča.

4.1.5 Konfiguracija in računalniški model

Načrtovali smo leteče krilo dolžine kril 180 cm in ga simulirali v programu Solidworks (slika 33). Tako smo dobili nazaj nagnjena konusna krila. Nagnjenost kril nazaj poveča statično stabilnost, saj premakne težišče nazaj in poveča razdaljo med njim in nevtralno stabilnostno točko. Konusna krila povečajo razmerje med dolžino in površino kril (angl. aspect ratio) in tako zmanjša težo, strukturne obremenitve, inducirani upor in poveča aerodinamično učinkovitost zaradi boljše eliptične porazdelitve vzgona, zmanjša pa površino krila. Na krila bi lahko namestili pregrade, ki preprečujejo zračni tok po dolžini krila. To bi preprečevalo zastoj celotnega krila pri visokih vpadnih kotih zračnega toka. Na koncu kril bi lahko dali horizontalna stabilizatorja, kar bi izboljšalo smerno stabilnost in zmanjšalo zračne vrtince na koncu kril in zmanjša upor. Zakrilci skrbita za usmerjanje letala.



Slika 33: Simulacija CAD modela v razvoju

4.2 Uporaba rešitve na modelu

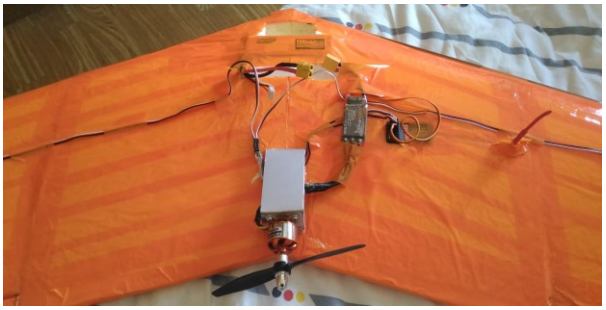
4.2.1 Izdelava

Dejanski model je zelo poenostavljen CAD model, ki je za testiranje naše rešitve povsem dovolj. Leteče krilo smo izdelali iz stiroporja (EPS), ker je redkejši in torej lažji od materiala XPS, čeprav tudi bolj krhek. Iz plošč stiroporja smo s pomočjo lesenih profilov in z vročo žico, ki ob priklopu na električni tok deluje kot upor, izrezali posamezne dele letečega krila. Ob sestavi smo ga ojačali s karbonskimi palčkami spodaj in znotraj tako, da se krilom zmanjša upogibanje, palčki spredaj pa delujeta kot odbijač pri nesreči. Letalo smo prelepili s posebnim 45nm debelim barvnim lepilnim trakom in ga tako ojačali. Zakrilci smo naredili iz deprona.

4.2.2 Izbira in integracija komponent

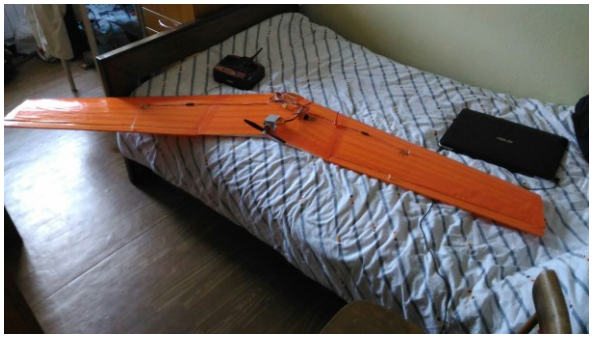
Pri izbiri komponent (slika 34.1) smo pazili na težo, enostavnost, ceno in primernost za leteče krilo naše velikosti. Vse komponente smo skušali čimbolj vgraditi v samo obliko letala. Razlogi za to so aerodinamična učinkovitost, estetika in lega težišča. Lega težišča naj bi bila približno 126.4 mm od zadnjega dela. Težišče preveč spredaj zmanjša učinkovitost letala, preveč zadaj pa njegovo stabilnost.

Na ustreznih mestih smo v stiropor izrezali vdobline, kamor smo namestili digitalna servo motorja in ju povezali z zakrilci (slika 34.2), LiPo baterijo, krmilnik ESC in radijski sprejemnik. Na koncu smo naredili nosilec za motor iz aluminija in ga na krilo prilepili z vročim lepilom. Namestili smo motor in na njega propeler. Samo letalo tehta 530g, naša rešitev tehta 200g, torej skupaj 730g, motor pa ima z nameščenim propelerjem okoli en kilogram potiska. Najprej smo letalo testirali z navadnim radijskim oddajnikom, potem pa še z našo rešitvijo. Slika 36 prikazuje uporabljeno in popravljeno letalo.



Slika 34.1: Integracija komponent

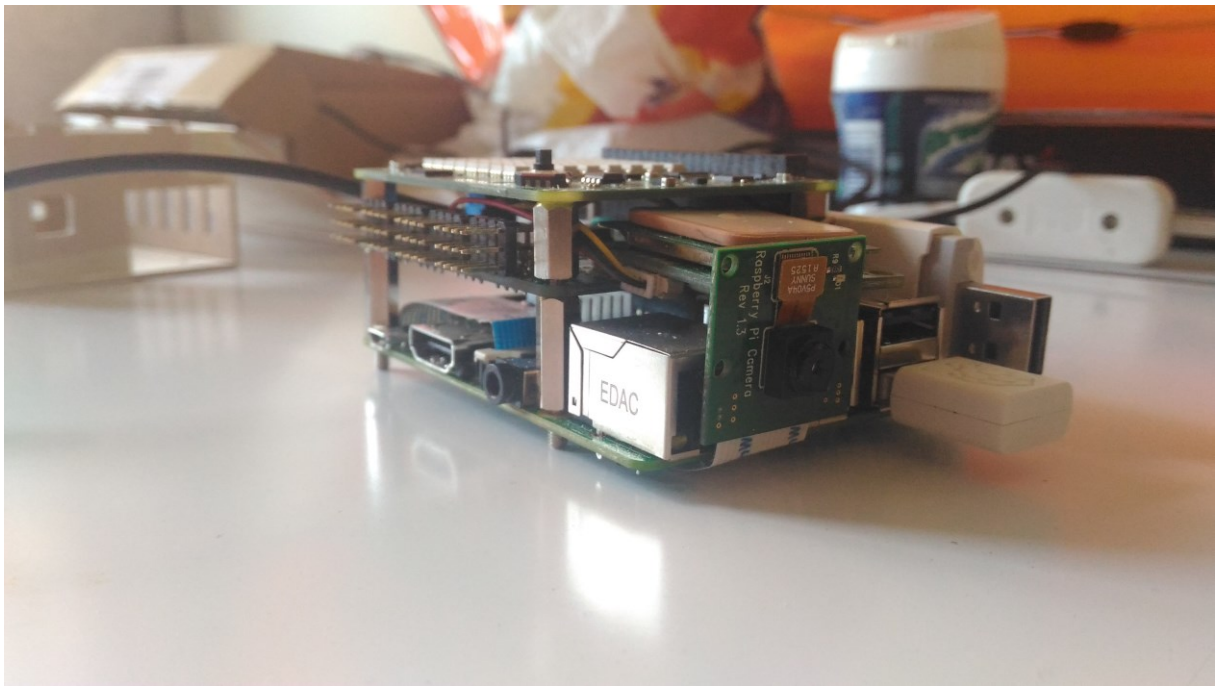
Slika 34.2: Montaža servo motorjev



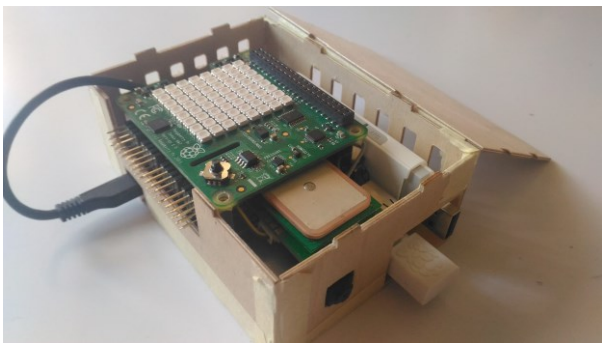
Slika 36: Opremljeno leteče krilo naše rešitve

4.2.3 Testiranje rešitve

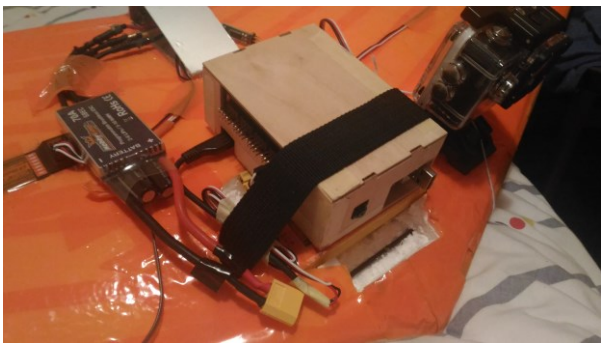
Preden smo rešitev lahko pritrdili na letalo, smo morali poskrbeti za kompaktnost izdelka (slika 37), sestavo z uporabo kovinskih vmesnikov med ploščami, prilagoditvami in postavitev v izdelano ohišje (slika 38). Rešitev se napaja iz enega vira, zato je potrebno paziti, da ne uporabimo premalo zmogljiv regulator napetosti v krmilniku ESC. Rešitev je dokaj kompaktna, še vedno pa je le prototip, saj ima predvsem elektrotehnični del rešitve še veliko prostora za izboljšave. Rešitev smo pripravili tako, da se vse zažene avtomatsko. To smo dosegli s pomočjo veliko truda in poskusi z uporabo programov za zakon grafičnih okolij in avtomatski zagon. Težave so nam povzročali hrošče v brskalniku Firefox, ampak smo na koncu vedno našli ustrezno rešitev.



Slika 37: Končni prototip rešitve



Slika 38: Rešitev v prototipnem ohišju



Slika 39: Rešitev v varnem a realnem okolju

Delovanje rešitve je bilo testirano v varnem okolju (slika 39) brez možnosti poškodb in nevarnih situacij. Rešitve trenutno še nismo uporabili v letu v naravi, saj zaradi pomanjkanja časa, odpravljanja napak, izrednih situacij in neugodnih vremenskih razmer za letenje to žal ni bilo mogoče. Testirali smo delovanje na tleh. Funkcionalnost rešitve je ustrezna, vedno obstajajo hrošči in napake, ki jih še nismo odkrili. Za dobro testirano rešitev potrebujemo razvijalce, testerje, uporabnike in mnogo časa, da omejimo nepravilno in neželeno delovanje prototipa in povečano zadovoljnost končnih uporabnikov.

4.2.4 Nadaljnji razvoj rešitve

Funkcionalnosti so bile določene že prej, dokončno pa smo večino tudi implementirali. Žal zaradi pomanjkanja časa nekaterih ni bilo mogoče dokončati.

4.2.4.1 Stabilizacija

Stabilizacijo rešitve smo sprogramirali s primerjavo vhodnih (nagib telefona) in izhodnih (nagib RPi) podatkov krmilnega sistema. Ogledali smo si osnove PID krmilnikov, zaradi enostavnosti in področja problema, s katerim se ukvarjamo pa smo implementirali samo P del, to pomeni proporcionalno krmiljenje: večja je napaka, večji je popravek. Tukaj bi bilo za dodelati, da se uporablja celoten PID krmilnik za večjo odpornost na stranske vplive in napake.

4.2.4.2 Avtopilot

Funkcionalnost avtopilota obsega obračanje, držanje smeri, držanje naklona in držanje višine. V obsegu diplomske naloge je v celoti implementirana samo funkcija držanja naklona. Manjka le gola logika preverjanja in klica funkcij.

4.2.4.3 Zemljevid in načrtovanje misij

Implementirana je bila osnova z zemljevidom, prikazom lokacije uporabnika in letala za nadaljno možno implementacijo načrtovanja misij. Delno je razvito tudi risanje poti in računanje razdalj in smeri za letalo. Nadaljni razvoj nima omejitev, samo ena od idej je recimo dostava poštnih paketov, kot to počne podjetje Amazon. S primerno vrsto brezpilotnega letala bi bilo možno sprogramirati rešitev, da z ukazi na zemljevidu omogočimo načrtovanje krajev vertikalnega vzleta, pristanka in ostalih akcij (odpiranje loput, spust tovora, pobiranje tovora, signaliziranje), kar bi pomenilo neomejen potencial, če državni zakoni dovoljujejo uporabo brezpilotnih letal na tak način. V Sloveniji je to trenutno nedovoljeno, v ZDA pa Amazon širi svoje dostavne sposobnosti z legalizacijo zakonov letalske agencije FAA, ki zakone ureja. Možnosti je ogromno, od dostave prve pomoči poškodovancem na težje dostopnem terenu, do nadzora kriminala ali poslikavi terena. Seveda bi bilo treba uporabiti kakšen senzor več, boljše algoritme in prevsem temeljiteje testirati rešitev, preden bi šla v dejansko uporabo. Veliko stvari že obstaja in deluje, vedno pa se najde kaj novega.

Poglavje 5 Tehnologija

Zelo na kratko bomo predstavili uporabljena programska razvijalna orodja, integrirana razvojna okolja, pripomočke in jezike. Pri tem se vidi, kaj je potrebno za razvoj sodobne programske opreme, ki temelji na več sistemih, platformah in programskih jezikih.

5.1 Programska razvijalna orodja

Uporabljali smo dve razvijalski orodji, čeprav sta navidezno združeni v eno. Xamarin Android in Android SDK delujeta odlično, vseeno pa imata tudi svoje pomanjkljivosti.

5.1.1 Xamarin Android in Android SDK

Xamarin [68] je programsko razvijalno orodje za sisteme Windows Phone, Android in iOS.

Njegov namen je ponovna uporaba kode pri razvoju za več mobilnih platform, saj omogoča, da aplikacijo napišemo enkrat, ta pa deluje na različnih sistemih. Za uporabo je na voljo IDE Xamarin Studio, obstaja pa tudi vmesnik za Visual Studio. Čeprav gre za razvoj za več

platform (angl. cross-platform), je delno treba pisati kodo za vsako platformo posebej, to pa zato, ker Xamarin zgradi matično (angl. native) aplikacijo. Če gre za Android razvoj, s sklici Jave zgradi datoteko `.apk` in omogoča enake zmogljivosti, kot če bi aplikacijo napisali v Javi. Zaradi tega je potrebno poznati vsako platformo, za katero razvijamo aplikacijo. Uporabljamo namreč matični uporabniški vmesnik in API-je. Podobno orodje je Codename One, samo da programiramo v Javi, medtem ko v Xamarinu programiramo v C#.

Xamarin pa je imel kaj nekaj slabosti, si jo jih deloma že odpravili (draga licenca okoli 2000$ letno, potrebno poznavanje ciljnih platform in strma učna krivulja zaradi pomanjkanja dokumentacije).

Njegove prednosti so delno povezave z njegovimi slabostmi, postal je namreč odprtokoden, izboljšali so tudi dokumentacijo, kljub različnim platformam omogoča programiranje v istem programskem jeziki (C#). Prav tako je zmogljivost aplikacij neokrnjena, kar je boljše od nekaterih konkurenčnih orodij.

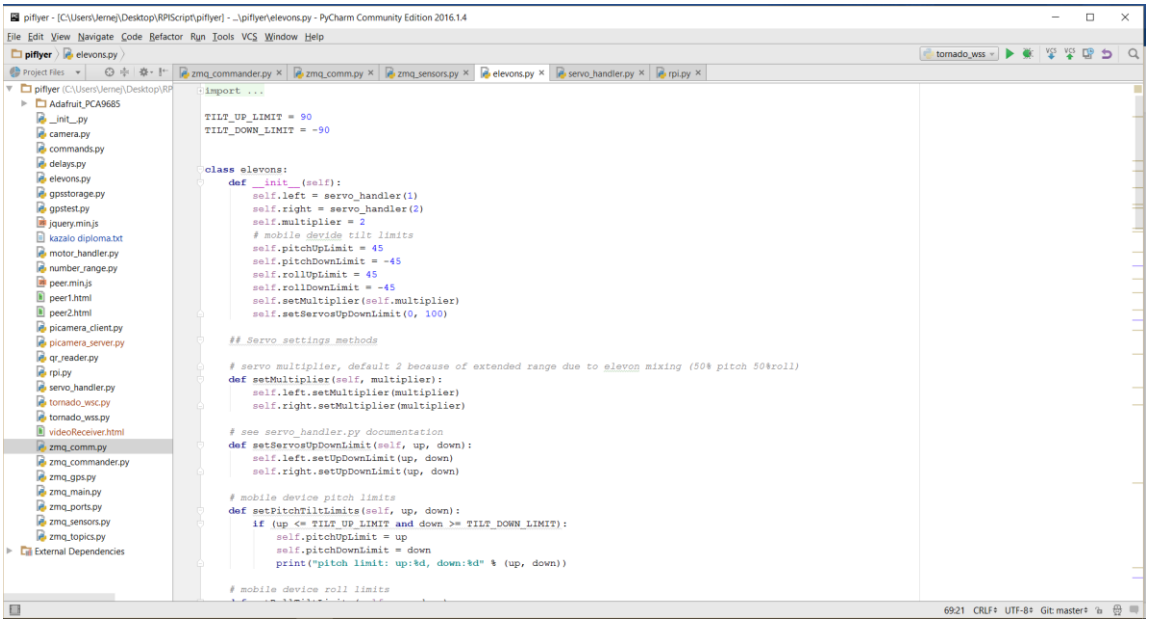
Knjižnica Xamarin uporablja Android SDK, če razvijamo za sistem Android.

5.2 Integrirana razvojna okolja

Za razvoj programske opreme smo uporabljali dva IDE-ja: python del smo programirali v IDE-ju Pycharm, Android del pa v Visual Studiu. Namesto Visual Studia bi lahko uporabljali tudi Xamarin Studio, vendar smo se zaradi poznanega okolja raje odločili za prvega.

5.2.1 Jetbrains Pycharm

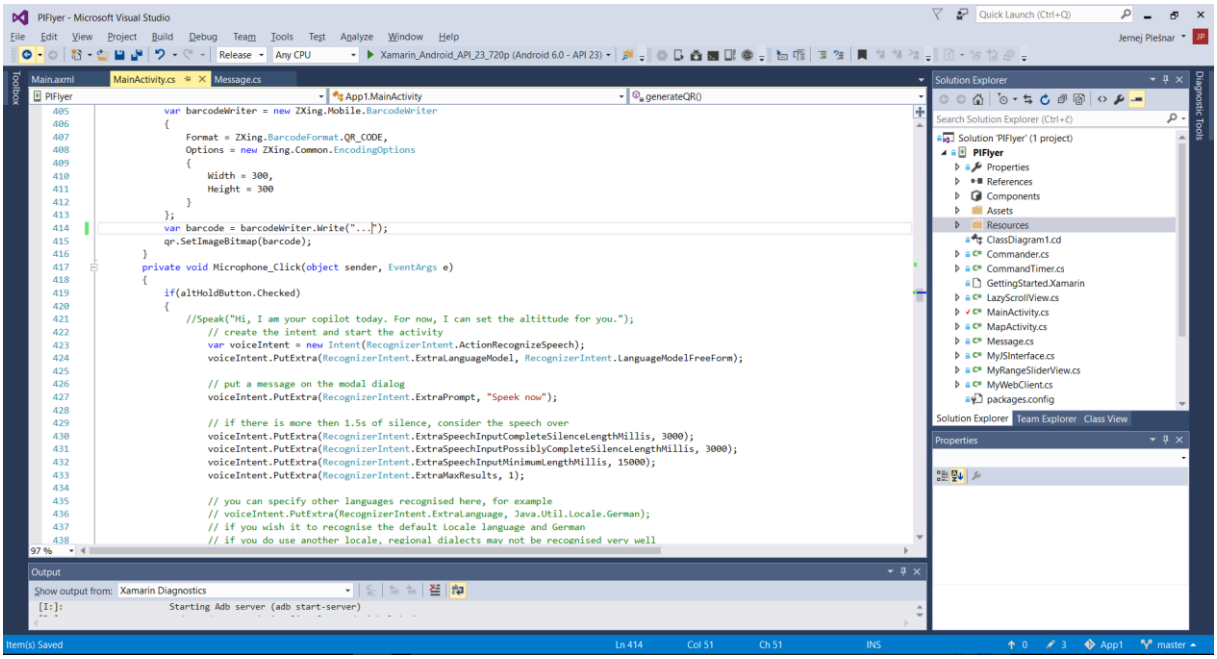
Jetbrains Pycharm Community 2016.1 (slika 39) [130] omogoča razvoj Python programov, podpira pa tudi spletni razvoj z vmesnikom Django. Ponuja analizo kode, razhroščevalnik, testna orodja, integracijo s sistemi za upravljanje verzij VCS (angl. Version Control) in inteligentno dopolnjevanje kode.



Slika 39: Razvojno orodje Pycharm Community

5.2.2 Microsoft Visual Studio

5.2.2 Microsoft Visual Studio Community 2015 (slika 40) [131] omogoča razvoj programov tako za namizne, spletne in mobilne aplikacije. Podpira pogone Node.js, .NET, Unity, Office itd. Urejevalnik besedil omogoča pisanje kode v C#, C++, F#, JSON, HTML, PHP, Python, TypeScript CSS, Sass, Less in več. Za vse jezike uporablja IntelliSense, močno orodje za dopolnjevanje kode. Z namestitvijo dodatnik SDK-jev in vtičnikov lahko programiramo tudi za Windows Phone, Android, iOS, Azure Cloud itd. Vsebuje močna orodja za prevajanje, razhroščevanje, upravljanje s paketi, testiranje programske opreme, analizo in preverjanje kakovosti kode, povezovanje s podatkovnimi bazami, oddaljenimi strežniki, vsebuje orodja za nadzor nad verzijami in delo v skupinah.

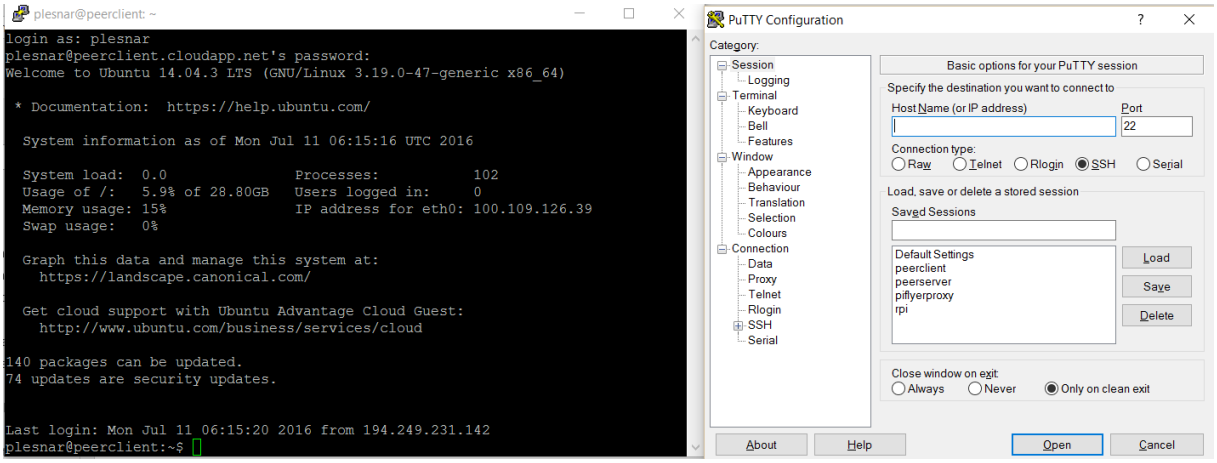


Slika 40: Razvojno orodje Visual Studio Community

5.3 Pripomočki

Pri razvoju smo uporabljali razna podporna orodja, ki niso razvijalska, a bi brez njih vseeno težko nemoteno razvijali rešitev. Uporabljali smo naslednja orodja:

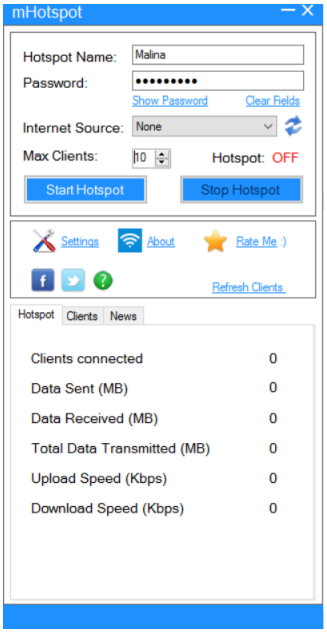
- Putty (slika 41) [134] - SSH in Telnet odjemalec za Windows in Unix sistem;



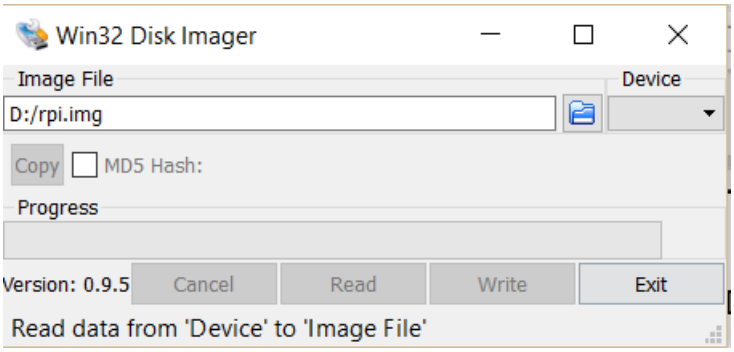
Slika 41: Pripomoček Putty

- mHotspot [135] - program, ki ustvari brezžično dostopno točko na Windows sistemu (slika 42). Potrebujemo le brezžično mrežno kartico, ki je v prenosnikih že privzeto;

- Win32DiskImager [136] - program za pisanje in branje pomnilniških slik ISO/IMG na/iz pomnilniških medijev (slika 43);

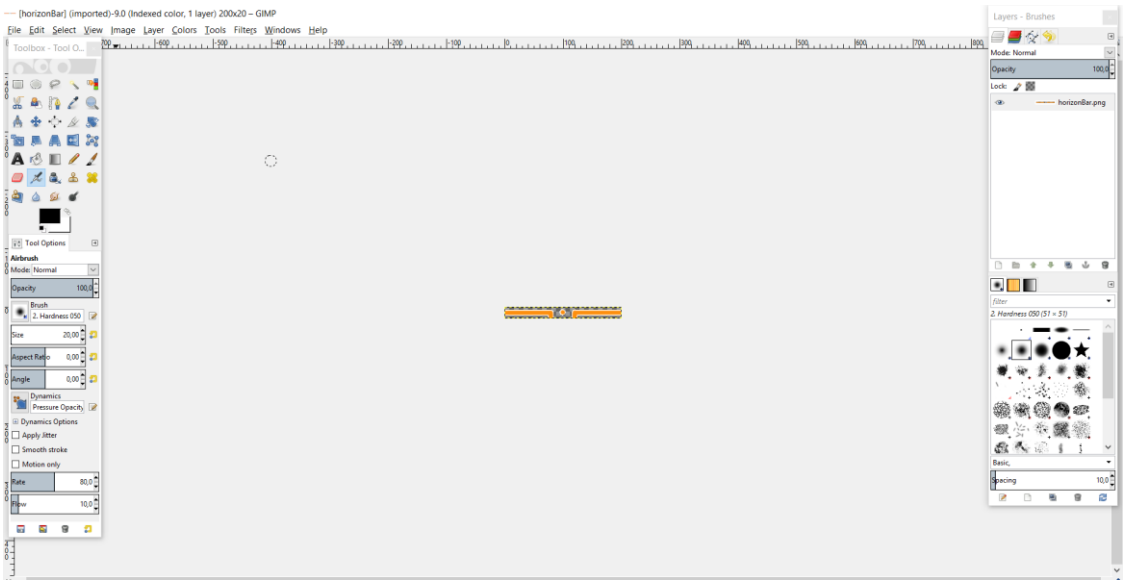


Slika 42: mHotstpot

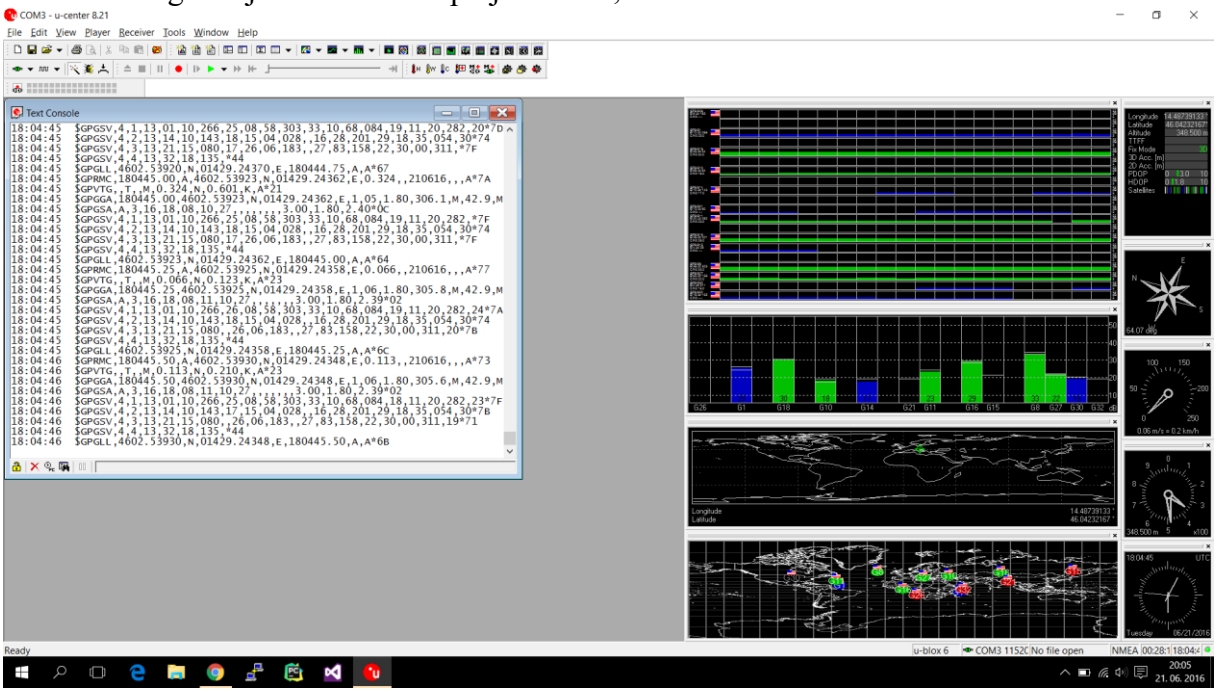


Slika 43: Win32DiskImager (desno)

- Gimp [137] - odprtokodno orodje za grafično oblikovanje, ki nam je omogočil risanje nekaterih grafičnih elementov za uporabniški vmesnik mobilne aplikacije (slika 44);

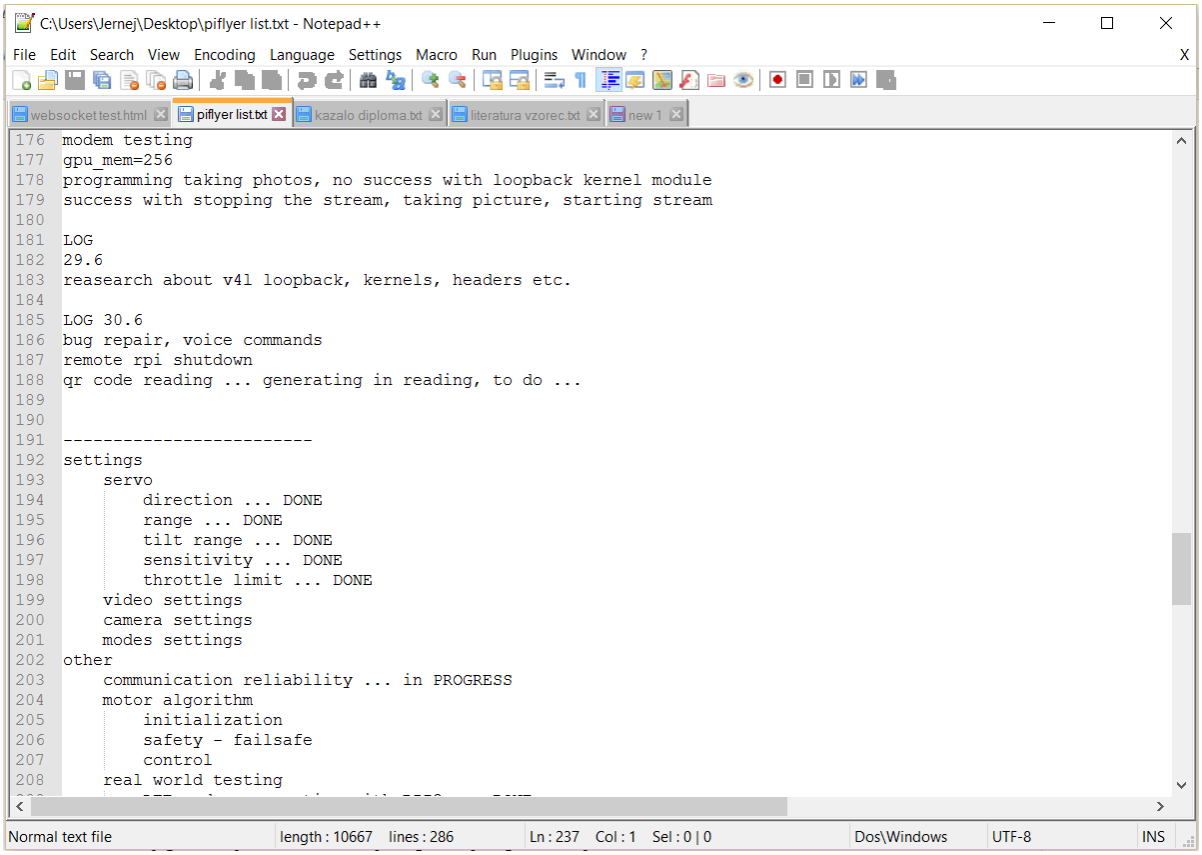

Slika 44: Orodje Gimp

- u-Center 8.21 [138] - programska rešitev (slika 45), namenjena za uporabo in konfiguracijo u-blox GPS sprejemnikov;



Slika 45: Programska oprema u-Center

- Notepad++ [139] - urejevalnik besedil in programske kode (slika 46), s katerim smo si pomagali pri urejanju spletne aplikacije, pisanju zapiskov za spremljanje razvoja, pisanju prvega osnutka besedila za diplomsko nalogo in načrtovanje funkcionalnosti, ki bi bile primerne v naši rešitvi.



Slika 46: Pripomoček Notepad++

5.4 Programski, skriptni in strukturni jeziki

Nekaj glavnih jezikov, ki smo jih uporabljali pri razvoju:

- C# smo uporabljali za programiranje Android aplikacije v Xamarin vmesniku;
- Python smo uporabljali za razvoj programske opreme, ki je namenjena za RPi;
- Bash - linuxova lupina, uporabljali smo jo za pisanje lahkih skript;
- Javascript, jquery - knjižnici namenjeni za odjemalčevo stran v spletni aplikaciji, uporabljeni sta bili za komunikacijski del sistema;
- HTML in CSS – strukturni in oblikovni jezik vsake spletne strani ali aplikacije.

5.5 Aplikacijski vmesniki in servisi

Pri razvoju smo uporabili nekaj aplikacijskih vmesnikov in servisov iz strani Androida in Googla. Na kratko bomo opisali njihov namen.

5.5.1 Servisi Google Play in Elevation, Maps API

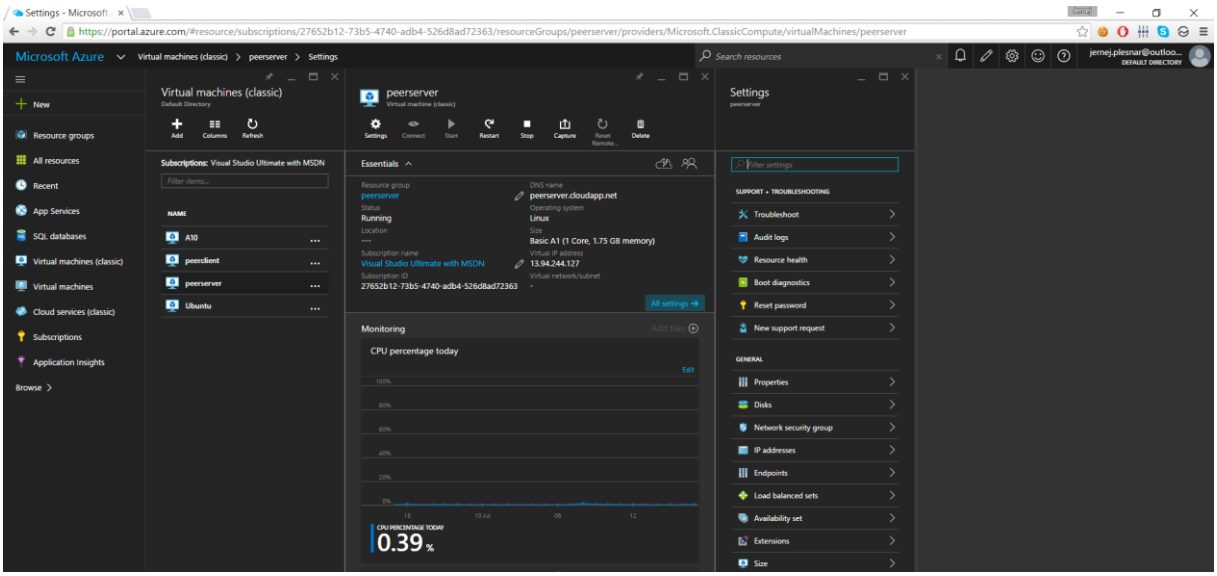
Storitev Google Play [140] nam omogoča dostop do Googlovih API-jev in funkcij (kot so Zemljividi in Google+). Google Maps API [141] je aplikacijski vmesnik, ki nam omogoča vgradnjo Googlovih zemljevidov (in vseh pripadajočih funkcionalnosti nad njimi) v svojo aplikacijo. Google elevation service [142] je ena od funkcionalnosti Google Maps API-ja, ki nam za katerokoli točko na zemljevidu vrne njeno nadmorsko višino.

5.5.2 Android Speech-to-text in Text-to-speech

Prvi omogoča pretvorbo govora v niz besed, drugi pa niza besed v govor. Obe funkcionalnosti sta del sistema Android. Pri tem nam pomaga vmesnik android.speech [143].

5.5.3 Oblačne storitve Microsoft Azure

Azurjev oblak [144] nam je s pomočjo licence MSP (angl. Microsoft Student Partner) omogočil najem treh virtualnih strežnikov z operacijskim sistemom Linux, brez katerih komunikacijski del ne bi deloval. S storitvijo upravljamo s pomočjo nadzorne plošče (slika 47).



Slika 47: Nadzorna plošča Microsoft Azure

Poglavje 6 Sklepne ugotovitve

V diplomski nalogi smo iz preproste ideje z malo znanja o Androidu in krmilniku Raspberry Pi in P2P komunikaciji podali na dolgo pot raziskovalno-razvojnega dela in zasnovali programsko-strojno rešitev, jo priredili in nadgradili. Sproti smo ugotavljali dodatne težave in iskali možni poti.

Pridobili smo ogromno računalniškega znanja, poleg tega pa nekaj poznavanja elektrotehnike, ponovili osnove aeronavtike in fizike na splošno, osvojili osnove CAD modeliranja in izdelave modelarskega modela letečega krila.

Razvili smo učinkovito rešitev, ki je sestavljeno iz:

- strojnega dela mikrokrmilnika Raspberry Pi 2, povezanega na sensorični in aktuatorski dodatek, kamero, wWi-Fi sprejemnik in 4G modem in celote postavljene v ohišje;
- programskega dela na RPi, katerega arhitektura temelji na medprocesni komunikaciji posameznih modulov, ki se izvajajo neodvisno vsak v svojem procesu, vključno s komunikacijo procesov z brskalnikom;
- komunikacijskega dela na osnovi P2P WebRtc komunikacije z osnovo na Linux TURN, spletnemu in signalizacijskemu strežniku na Azure platformi in spletne aplikacije s knjižnico PeerJS, ki jo uporablja Android aplikacija in spletni brskalnik na RPi ter
- Android aplikacije razvite v Xamarin platformi kot uporabniški vmesnik med rešitvijo in človekom.

Nekaj stvari nam zaradi časovnih omejitev žal ni uspelo izdelati do konca: povezava GPS sprejemnika preko serijskega vmesnika (namesto USB), funkcionalnosti avtopilota, testiranja celotne programske opreme, odprave vseh napak, implementacija govornega asistenta Android v celoti, ohišja iz pleksi stekla in poudarka na estetiki strojne rešitve.

Med delom so nas presenetile številne težave, zapleti in spremembe (tako pri sami izdelavi diplomske naloge), kot tudi osebne preizkušnje, kot je smrt enega od staršev. Za izdelavo pisnega izdelka smo porabili okoli 5-10% celotnega časa. Največ dela zahteva pot od ideje do delujoče rešitve, največji del od tega pa je šlo za razvoj komunikacije, kar je tudi zahtevalo največ skrbi, preizkušenj in odpravljanja težav.

Naša ocena porabe časa za celotno diplomsko delo je okoli 2600 ur, od česar je bilo okoli 200 ur opravljenih med študijem, prakso in športno aktivnostjo od aprila 2015 do septembra 2015, okoli 1000 ur med oktobrom 2015 in majem 2016 med rednim delom in športno aktivnostjo, okoli 1100 ur junija in julija 2016 in okoli 300 ur avgusta in septembra 2016.

Glavni prispevki diplomske naloge so raziskava, načrtovanje in razvoj na področju IoT, P2P in medprocesne komunikacije, WebRTC protokola in sorodnih knjižnic, povezav različnih tehnologij in sistemov, mikrokrmilnika Raspberry Pi, sistema Android za krmiljenje brezpilotnih letal, uporabe Googlovih in Android storitev in Linux strežnikov za robotski sistem. Kot velik prispevek vidimo tudi opis poteka razvoja nove programsko-strojne rešitve v diplomski nalogi.

Ti prispevki so pomembni, saj prikazujejo, da je Raspberry Pi zrel za uporabo tudi v industriji ter pomembnost mobilnih omrežij v tehnologiji. Tudi WebRTC je pomemben v praksi. Skozi potek diplomske naloge le površno vidimo količino težav in zapletov pri razvoju povsem nove rešitve, ki drugje še ne obstaja, količino truda in pomembnost modularnosti posameznih delov rešitve, vztrajnosti, poenostavljanja in celostnega pogleda načrtovalca in razvijalca.

Možna uporaba rešitve je na modelarskem področju, saj je namenjena preprostemu uporabniku, kljub temu pa je (s svojimi možnosti razširitve) možna uporaba za robotske sisteme različnih vrst: od krmiljenja tople grede, robotske roke, podmornice, ladje, avtomobila do letala in rakete ne glede na to, če se ta nahaja na drugem koncu sveta, dokler ima le dostop do interneta.

Možnih izboljšav in razširitev je ogromno. Za začetek bi bilo za dodelati manjkajočih funkcionalnosti avtopilota in načrtovanja misij, s katerima bi bilo možno še enostavnejše letenje na dolge razdalje brez stalnega držanja telefona v roki in preprosto uporabo glasovnih ukazov. S pomočjo podatkov o višini letala in terena iz Googlove storitve bi bilo možno varno leteti ne glede na spremembo terena in izogibanje hribom, pa tudi opozarjanje in obveščanje o dogodkih. Uporabe računalniškega vida in dodatnih senzorjev nam bi omogočala prepoznavanje ovir, nadzor prometa, požarov, merjenja koncentracij nevarnih plinov in kakovosti zraka. Upravljanje več robotov na različnih koncih sveta in povezavo v internetni oblak ni več nedosegljivo. S pomočjo internetne povezave so možnosti neomejene. Za delovanje rešitve je potrebno tudi namestiti ustrezne knjižnice in programsko opremo [1] na Raspberry Pi in konfigurirati Raspberry Pi. Poleg tega je potrebna strojna oprema, ki podpira delovanje programske opreme in konfiguracije.

[1] Do celotne izvorne kode v programskem jeziku Python lahko dostopate na naslovu https://github.com/jeryfast/piflyer/tree/master/piflyer, koda Android aplikacije pa ni objavljena na internetu.

Literatura

[1] Edx certificate [Online]. Dosegljivo:\\ https://s3.amazonaws.com/verify.edx.org/downloads/60b6addb529c4bce952d89867e450f6e/Certificate.pdf. [Dostopno 11. 7. 2016].

[2] DBF [Online]. Dosegljivo:\\ http://www.aiaadbf.org/. [Dostopno 11. 7. 2016].

[3] Carbon flyer [Online]. Dosegljivo:\\ https://www.indiegogo.com/projects/carbon-flyer-cell-phone-controlled-plane#/. [Dostopno 11. 7. 2016].

[4] PowerUp FPV [Online]. Dosegljivo:\\ https://www.kickstarter.com/projects/393053146/powerup-fpv-live-streaming-paper-airplane-drone/description. [Dostopno 11. 7. 2016].

[5] Pixhawk [Online]. Dosegljivo:\\ https://pixhawk.org/. [Dostopno 11. 7. 2016].

[6] 4gmetry [Online]. Dosegljivo:\\ http://4gmetry.voltarobots.com/. [Dostopno 11. 7. 2016].

[7] C-Astral [Online]. Dosegljivo:\\ http://www.c-astral.com/. [Dostopno 11. 7. 2016].

[8] Emlid [Online]. Dosegljivo:\\ https://emlid.com/. [Dostopno 11. 7. 2016].

[9] Erlebrain [Online]. Dosegljivo:\\ https://erlerobotics.com/blog/product/erle-brain-v2/. [Dostopno 11. 7. 2016].

[10] Micro drone 3.0 [Online]. Dosegljivo:\\ https://www.indiegogo.com/projects/micro-drone-3-0-flight-in-the-palm-of-your-hand--2#/. [Dostopno 11. 7. 2016].

[11] Parrot ARDrone [Online]. Dosegljivo:\\ http://www.parrot.com/fr/produits/ardrone-2/. [Dostopno 11. 7. 2016].

[12] Skydrone [Online]. Dosegljivo:\\ http://www.skydrone.aero/fpv. [Dostopno 11. 7. 2016].

[13] Mobilna aplikacija za deljenja omrežja [Online]. Dosegljivo:\\ http://www.rcgroups.com/forums/showthread.php?t=2301482. [Dostopno 11. 7. 2016].

[14] Prenos slike s pomočo RPi [Online]. Dosegljivo:\\ http://diydrones.com/profiles/blogs/fpv-setup-with-raspberry-pi. [Dostopno 11. 7. 2016].

[15] Telemetrija s pomočjo usmerjevalnika in RPi [Online]. Dosegljivo:\\ http://diydrones.com/m/blogpost?id=705844%3ABlogPost%3A1234094. [Dostopno 11. 7. 2016].

[16] Ideja v članku [Online]. Dosegljivo:\\ http://uavmatrix.com/Blog/56. [Dostopno 11. 7. 2016].

[17] Slice [Online]. Dosegljivo:\\ https://www.kickstarter.com/projects/fiveninjas/slice-a-media-player-and-more/description. [Dostopno 11. 7. 2016].

[18] Wildlife-cam-kit [Online]. Dosegljivo:\\ https://www.kickstarter.com/projects/279364224/naturebytes-wildlife-cam-kit-digital-making-for-wi/description. [Dostopno 11. 7. 2016].

[19] Simobil pokritje [Online]. Dosegljivo:\\ https://www.simobil.si/omrezje/zemljevid-pokritosti. [Dostopno 11. 7. 2016].

[20] Mobitel pokritje [Online]. Dosegljivo:\\ http://www.telekom.si/pomoc-in-podpora/teme-pomoci/pokritost-in-dostopnost/pokritost-mobilnega-omrezja. [Dostopno 11. 7. 2016].

[21] Prihod 5G [Online]. Dosegljivo:\\ http://www.delo.si/znanje/infoteh/5g-mobilna-omrezja-prihodnosti-sele-dobivajo-obrise.html. [Dostopno 11. 7. 2016].

[22] Latenca 5G [Online]. Dosegljivo:\\ http://www.huawei.com/minisite/5g/en/defining-5g.html. [Dostopno 11. 7. 2016].

[23] Arduino [Online]. Dosegljivo:\\ https://www.arduino.cc/. [Dostopno 11. 7. 2016].

[24] Texas Instruments [Online]. Dosegljivo:\\ http://www.ti.com/. [Dostopno 11. 7. 2016].

[25] Microchip PIC [Online]. Dosegljivo:\\ http://www.microchip.com/design-centers/microcontrollers. [Dostopno 11. 7. 2016].

[26] STM [Online]. Dosegljivo:\\ http://www.st.com/content/st_com/en.html. [Dostopno 11. 7. 2016].

[27] Raspberry Pi [Online]. Dosegljivo:\\ https://www.raspberrypi.org/. [Dostopno 11. 7. 2016].

[28] Galileo [Online]. Dosegljivo:\\ http://www.intel.com/content/www/us/en/do-it-yourself/galileo-maker-quark-board.html. [Dostopno 11. 7. 2016].

[29] Beaglebone [Online]. Dosegljivo:\\ http://beagleboard.org/bone. [Dostopno 11. 7. 2016].

[30] Odroid [Online]. Dosegljivo:\\ http://www.hardkernel.com/main/main.php. [Dostopno 11. 7. 2016].

[31] FY Panda II [Online]. Dosegljivo:\\ http://www.feiyu-tech.com/products/11/. [Dostopno 11. 7. 2016].

[32] DJI Naza [Online]. Dosegljivo:\\ http://www.dji.com/product/naza-m-v2. [Dostopno 11. 7. 2016].

[33] Piccolo [Online]. Dosegljivo:\\ http://www.cloudcaptech.com/products/auto-pilots. [Dostopno 11. 7. 2016].

[34] HSE [Online]. Dosegljivo:\\ http://www.hse-uav.com/. [Dostopno 11. 7. 2016].

[35] Zanesljivost strežnikov [Online]. Dosegljivo:\\ http://www.cloudberrylab.com/blog/azure-vm-vs-amazon-ec2-vs-google-ce-cloud-computing-comparison/. [Dostopno 11. 7. 2016].

[36] Raspberry Pi [Online]. Dosegljivo:\\ https://www.adafruit.com/products/2358. [Dostopno 11. 7. 2016].

[37] Servo HAT [Online]. Dosegljivo:\\ https://www.adafruit.com/product/2327. [Dostopno 11. 7. 2016].

[38] Sense HAT [Online]. Dosegljivo:\\ https://www.adafruit.com/products/2738. [Dostopno 11. 7. 2016].

[39] PiCamera [Online]. Dosegljivo:\\ https://www.adafruit.com/products/1367. [Dostopno 11. 7. 2016].

[40] Wi-Fi mrežna kartica [Online]. Dosegljivo:\\ https://www.adafruit.com/products/2638. [Dostopno 11. 7. 2016].

[41] LTE mrežna kartica [Online]. Dosegljivo:\\ http://consumer.huawei.com/en/mobile-broadband/dongles/tech-specs/e3372.htm. [Dostopno 11. 7. 2016].

[42] GPS [Online]. Dosegljivo:\\ http://www.hobbyking.com/hobbyking/store/__69985__NEO_6M_GPS_Module_RU_Warehouse_.html. [Dostopno 11. 7. 2016].

[43] ESC in UBEC [Online]. Dosegljivo:\\ http://www.4-max.co.uk/pdf/WhyDoINeedaUBEC.pdf. [Dostopno 11. 7. 2016].

[44] Napajalnik [Online]. Dosegljivo:\\ http://www.galagomarket.com/index.php/item/display/368/863_adapters_3d-systems_stontronics---raspberry-pi,-5v,-2a,-euro-uk---t5582dv---psu. [Dostopno 11. 7. 2016].

[45] VNC [Online]. Dosegljivo:\\ https://www.raspberrypi.org/documentation/remote-access/vnc/. [Dostopno 11. 7. 2016].

[46] X11 [Online]. Dosegljivo:\\ http://elinux.org/RPi_Remote_Access. [Dostopno 11. 7. 2016].

[47] YAGNI [Online]. Dosegljivo:\\ https://en.wikipedia.org/wiki/You_aren%27t_gonna_need_it. [Dostopno 11. 7. 2016].

[48] GIL [Online]. Dosegljivo:\\ https://wiki.python.org/moin/GlobalInterpreterLock. [Dostopno 11. 7. 2016].

[49] Multiprocessing [Online]. Dosegljivo:\\ https://pymotw.com/2/multiprocessing/basics.html. [Dostopno 11. 7. 2016].

[50] Pickle [Online]. Dosegljivo:\\ https://docs.python.org/2/library/pickle.html. [Dostopno 11. 7. 2016].

[51] Autobahn [Online]. Dosegljivo:\\ http://autobahn.ws/. [Dostopno 11. 7. 2016].

[52] Easy Websocket [Online]. Dosegljivo:\\ https://github.com/jeromeetienne/EasyWebsocket. [Dostopno 11. 7. 2016].

[53] Websockify [Online]. Dosegljivo:\\ https://github.com/kanaka/websockify. [Dostopno 11. 7. 2016].

[54] Simplewebsocket [Online]. Dosegljivo:\\ https://github.com/dpallot/simple-websocket-server. [Dostopno 11. 7. 2016].

[55] Tornado [Online]. Dosegljivo:\\ http://www.tornadoweb.org/en/stable/. [Dostopno 11. 7. 2016].

[56] ZeroMQ [Online]. Dosegljivo:\\ http://zeromq.org/community. [Dostopno 11. 7. 2016].

[57] IPC [Online]. Dosegljivo:\\ https://en.wikipedia.org/wiki/Inter-process_communication. [Dostopno 11. 7. 2016].

[58] Publish-subscribe [Online]. Dosegljivo:\\ http://zguide.zeromq.org/page:all. [Dostopno 11. 7. 2016].

[59] Latency [Online]. Dosegljivo:\\ http://zeromq.org/results:0mq-tests-v03. [Dostopno 11. 7. 2016].

[60] Connection RPi to GPS [Online]. Dosegljivo:\\ https://bigdanzblog.wordpress.com/2015/01/18/connecting-u-blox-neo-6m-GPS-to-raspberry-pi/. [Dostopno 11. 7. 2016].

[61] GPSd [Online]. Dosegljivo:\\ http://www.catb.org/GPSd/GPSd-time-service-howto.html. [Dostopno 11. 7. 2016].

[62] GPSd for python [Online]. Dosegljivo:\\ http://www.danmandle.com/blog/getting-GPSd-to-work-with-python/. [Dostopno 11. 7. 2016].

[63] Sakis3G [Online]. Dosegljivo:\\ http://www.sakis3g.com/. [Dostopno 11. 7. 2016].

[64] Holding mobile devices use cases [Online]. Dosegljivo:\\ http://www.uxmatters.com/mt/archives/2013/02/how-do-users-really-hold-mobile-devices.php. [Dostopno 11. 7. 2016].

[65] Holding mobile devices for different purposes [Online]. Dosegljivo:\\ http://www.yorku.ca/mack/ie2014.html. [Dostopno 11. 7. 2016].

[66] Holding mobile devices - mobile phones [Online]. Dosegljivo:\\ http://www.uxmatters.com/mt/archives/2013/02/images/HoldPhones_Figure-4.png. [Dostopno 11. 7. 2016].

[67] Holding mobile devices – tablets [Online]. Dosegljivo:\\ http://www.uxmatters.com/mt/archives/2013/02/images/HoldPhones_Figure-4.png. [Dostopno 11. 7. 2016].

[68] Xamarin [Online]. Dosegljivo:\\ https://www.xamarin.com/platform. [Dostopno 11. 7. 2016].

[69] Google Maps [Online]. Dosegljivo:\\ https://developers.google.com/maps/. [Dostopno 11. 7. 2016].

[70] SOCKS [Online]. Dosegljivo:\\ https://en.wikipedia.org/wiki/SOCKS. [Dostopno 11. 7. 2016].

[71] Proxy server [Online]. Dosegljivo:\\ https://en.wikipedia.org/wiki/Proxy_server. [Dostopno 11. 7. 2016].

[72] WebRTC [Online]. Dosegljivo:\\ http://www.html5rocks.com/en/tutorials/webrtc/basics/. [Dostopno 11. 7. 2016].

[73] WebRTC arhitecture [Online]. Dosegljivo:\\ https://webrtc.org/architecture/. [Dostopno 11. 7. 2016].

[74] STUN [Online]. Dosegljivo:\\ https://en.wikipedia.org/wiki/STUN. [Dostopno 11. 7. 2016].

[75] TURN [Online]. Dosegljivo:\\ https://en.wikipedia.org/wiki/Traversal_Using_Relays_around_NAT. [Dostopno 11. 7. 2016].

[76] ICE [Online]. Dosegljivo:\\ https://en.wikipedia.org/wiki/Interactive_Connectivity_Establishment. [Dostopno 11. 7. 2016].

[77] Socket.io [Online]. Dosegljivo:\\ http://socket.io/. [Dostopno 11. 7. 2016].

[78] PeerJS [Online]. Dosegljivo:\\ http://peerjs.com/. [Dostopno 11. 7. 2016].

[79] SkylinkJS [Online]. Dosegljivo:\\ https://github.com/Temasys/SkylinkJS. [Dostopno 11. 7. 2016].

[80] EasyRTC [Online]. Dosegljivo:\\ https://easyrtc.com/. [Dostopno 11. 7. 2016].

[81] SimpleWebRTC [Online]. Dosegljivo:\\ https://simplewebrtc.com/. [Dostopno 11. 7. 2016].

[82] Rtc-everywhere [Online]. Dosegljivo:\\ https://github.com/contra/rtc-everywhere. [Dostopno 11. 7. 2016].

[83] OpenTokRTC [Online]. Dosegljivo:\\ https://opentokrtc.com/. [Dostopno 11. 7. 2016].

[84] Talki.io [Online]. Dosegljivo:\\ https://talky.io/. [Dostopno 11. 7. 2016].

[85] AppRTC [Online]. Dosegljivo:\\ https://apprtc.appspot.com/. [Dostopno 11. 7. 2016].

[86] Uber [Online]. Dosegljivo:\\ https://www.uber.com/. [Dostopno 11. 7. 2016].

[87] Twillio [Online]. Dosegljivo:\\ https://www.twilio.com/. [Dostopno 11. 7. 2016].

[88] Turnserver [Online]. Dosegljivo:\\ https://github.com/jitsi/turnserver. [Dostopno 11. 7. 2016].

[89] Pysip [Online]. Dosegljivo:\\ http://www.pjsip.org/pjnath/docs/html/. [Dostopno 11. 7. 2016].

[90] Restund [Online]. Dosegljivo:\\ http://www.creytiv.com/restund.html. [Dostopno 11. 7. 2016].

[91] Creytiv [Online]. Dosegljivo:\\ http://creytiv.com/re.html. [Dostopno 11. 7. 2016].

[92] RFC5766-turn-server [Online]. Dosegljivo:\\ https://github.com/coturn/rfc5766-turn-server/. [Dostopno 11. 7. 2016].

[93] Coturn [Online]. Dosegljivo:\\ https://github.com/coturn/coturn. [Dostopno 11. 7. 2016].

[94] Xirsys [Online]. Dosegljivo:\\ https://xirsys.com/. [Dostopno 11. 7. 2016].

[95] ZeroTier [Online]. Dosegljivo:\\ https://www.zerotier.com/. [Dostopno 11. 7. 2016].

[96] Anyfirewall [Online]. Dosegljivo:\\ http://www.anyfirewall.com/. [Dostopno 11. 7. 2016].

[97] Estos [Online]. Dosegljivo:\\ https://www.estos.com/. [Dostopno 11. 7. 2016].

[98] Icelink [Online]. Dosegljivo:\\ http://www.icelink.co/. [Dostopno 11. 7. 2016].

[99] Trickle-ICE [Online]. Dosegljivo:\\ https://webrtc.github.io/samples/src/content/peerconnection/trickle-ice/. [Dostopno 11. 7. 2016].

[100] Viagenie turn server [Online]. Dosegljivo:\\ http://numb.viagenie.ca/. [Dostopno 11. 7. 2016].

[101] Peerjs issue [Online]. Dosegljivo:\\ https://github.com/peers/peerjs/issues/108. [Dostopno 11. 7. 2016].

[102] Browser interopability for WebRTC [Online]. Dosegljivo:\\ http://iswebrtcreadyyet.com/. [Dostopno 11. 7. 2016].

[103] SkyWay [Online]. Dosegljivo:\\ https://nttcom.github.io/skyway/en/docs/. [Dostopno 11. 7. 2016].

[104] BananaPI [Online]. Dosegljivo:\\ http://www.bananapi.org/. [Dostopno 11. 7. 2016].

[105] ArduinoYun [Online]. Dosegljivo:\\ https://www.arduino.cc/en/Main/ArduinoBoardYun. [Dostopno 11. 7. 2016].

[106] Crosswalk [Online]. Dosegljivo:\\ https://crosswalk-project.org. [Dostopno 11. 7. 2016].

[107] Dillo [Online]. Dosegljivo:\\ http://www.dillo.org/. [Dostopno 11. 7. 2016].

[108] Epiphany Web Browser [Online]. Dosegljivo:\\ https://wiki.gnome.org/Apps/Web. [Dostopno 11. 7. 2016].

[109] NetSurfWebBrowser [Online]. Dosegljivo:\\ http://www.netsurf-browser.org/. [Dostopno 11. 7. 2016].

[110] Chromium Web Browser [Online]. Dosegljivo:\\ https://www.chromium.org/. [Dostopno 11. 7. 2016].

[111] Iceweasel [Online]. Dosegljivo:\\ https://wiki.debian.org/Iceweasel. [Dostopno 11. 7. 2016].

[112] PyQt4 [Online]. Dosegljivo:\\ https://www.riverbankcomputing.com/software/pyqt/download. [Dostopno 11. 7. 2016].

[113] PySide [Online]. Dosegljivo:\\ https://wiki.qt.io/PySide. [Dostopno 11. 7. 2016].

[114] PyExecJS [Online]. Dosegljivo:\\ https://pypi.python.org/pypi/PyExecJS. [Dostopno 11. 7. 2016].

[115] PyJS [Online]. Dosegljivo:\\ http://pyjs.org/. [Dostopno 11. 7. 2016].

[116] Selenium [Online]. Dosegljivo:\\ https://pypi.python.org/pypi/selenium. [Dostopno 11. 7. 2016].

[117] GStreamer [Online]. Dosegljivo:\\ https://gstreamer.freedesktop.org/. [Dostopno 11. 7. 2016].

[118] Netcat [Online]. Dosegljivo:\\ https://en.wikipedia.org/wiki/Netcat. [Dostopno 11. 7. 2016].

[119] Raspberry Pi video stream options [Online]. Dosegljivo:\\ http://wiki.oz9aec.net/index.php/Raspberry_Pi_Camera. [Dostopno 11. 7. 2016].

[120] Kurento [Online]. Dosegljivo:\\ https://www.kurento.org/. [Dostopno 11. 7. 2016].

[121] Exagear desktop [Online]. Dosegljivo:\\ https://eltechs.com/product/exagear-desktop/. [Dostopno 11. 7. 2016].

[122] Uv4l [Online]. Dosegljivo:\\ http://www.linux-projects.org/documentation/uv4l-server/. [Dostopno 11. 7. 2016].

[123] RPi-Cam-Web-Interface [Online]. Dosegljivo:\\ http://elinux.org/RPi-Cam-Web-Interface. [Dostopno 11. 7. 2016].

[124] V4l2 [Online]. Dosegljivo:\\ https://en.wikipedia.org/wiki/Video4Linux. [Dostopno 11. 7. 2016].

[125] WebM [Online]. Dosegljivo:\\ http://blog.webmproject.org/2010/07/inside-webm-technology-vp8-intra-and.html. [Dostopno 11. 7. 2016].

[126] Introduction to Aeronautical Engineering course [Online]. Dosegljivo:\\ https://courses.edx.org/courses/DelftX/AE.1110x/3T2014/info. [Dostopno 11. 7. 2016].

[127] Bernoulli`s principle [Online]. Dosegljivo:\\ https://en.wikipedia.org/wiki/Bernoulli%27s_principle. Dostopno 11. 7. 2016].

[128] Flow simulation [Online]. Dosegljivo:\\ https://www.solidworks.com/sw/products/simulation/flow-simulation.htm. [Dostopno 11. 7. 2016].

[129] Parametric study [Online]. Dosegljivo:\\ https://www.solidworks.com/sw/products/simulation/flow-parametric-study.htm. [Dostopno 11. 7. 2016].

[130] Pycharm [Online]. Dosegljivo:\\ https://www.jetbrains.com/pycharm/. [Dostopno 11. 7. 2016].

[131] Visual studio [Online]. Dosegljivo:\\ https://www.visualstudio.com/products/vs-2015-product-editions. [Dostopno 11. 7. 2016].

[132] Skulpt [Online]. Dosegljivo:\\ http://www.skulpt.org/. [Dostopno 11. 7. 2016].

[133] Brython [Online]. Dosegljivo:\\ http://www.brython.info/. [Dostopno 11. 7. 2016].

[134] Putty [Online]. Dosegljivo:\\ http://www.chiark.greenend.org.uk/~sgtatham/putty/. [Dostopno 11. 7. 2016].

[135] mHotSpot [Online]. Dosegljivo:\\ http://www.mhotspot.com/. [Dostopno 11. 7. 2016].

[136] Win32DiskImager [Online]. Dosegljivo:\\ https://sourceforge.net/projects/win32diskimager/. [Dostopno 11. 7. 2016].

[137] Gimp [Online]. Dosegljivo:\\ https://www.gimp.org/. [Dostopno 11. 7. 2016].

[138] U-center [Online]. Dosegljivo:\\ https://www.u-blox.com/en/product/u-center-windows. [Dostopno 11. 7. 2016].

[139] Notepad++ [Online]. Dosegljivo:\\ https://notepad-plus-plus.org/. [Dostopno 11. 7. 2016].

[140] Android [Online]. Dosegljivo:\\ https://developers.google.com/android/guides/overview. [Dostopno 11. 7. 2016].

[141] Google Maps [Online]. Dosegljivo:\\ https://developers.google.com/maps/. [Dostopno 11. 7. 2016].

[142] Google Elevation Services [Online]. Dosegljivo:\\ https://developers.google.com/maps/documentation/elevation/intro. [Dostopno 11. 7. 2016].

[143] Android speech [Online]. Dosegljivo:\\ https://developer.android.com/reference/android/speech/package-summary.html. [Dostopno 11. 7. 2016].

[144] Microsoft Azure [Online]. Dosegljivo:\\ https://azure.microsoft.com/en-us/. [Dostopno 11. 7. 2016].

[145] Pixhawk wiring [Online]. Dosegljivo:\\ http://ardupilot.org/copter/docs/advanced-pixhawk-quadcopter-wiring-chart.html. [Dostopno 11. 7. 2016].

[146] UAV Component wiring [Online]. Dosegljivo:\\ http://www.rcgroups.com/forums/showthread.php?t=1286603&page=2. [Dostopno 11. 7. 2016].

[147] IP [Online]. Dosegljivo:\\ https://en.wikipedia.org/wiki/Internet_protocol_suite [Dostopno 11. 7. 2016].

[148] Use of microcontrollers [Online]. Dosegljivo:\\ https://www.toptal.com/c/how-i-made-a-fully-functional-arduino-weather-station-for-300 [Dostopno 11. 7. 2016].

[149] Mobile devices users statistics [Online]. Dosegljivo:\\ https://www.novelucent.com/services/mcommerce [Dostopno 11. 7. 2016].

[150] v4l2-ctl [Online]. Dosegljivo:\\ https://www.mankier.com/1/v4l2-ctl [Dostopno 11. 7. 2016].

[151] Devices behind symmetric NAT [Online]. Dosegljivo:\\ https://www.researchgate.net/figure/228977939_fig1_Figure-3-Frequency-distribution-of-NAT-timeout-for-UDP [Dostopno 11. 7. 2016].

[152] HS522 [Online]. Dosegljivo:\\ https://www.aerodesign.de/english/profile/profile_s.htm [Dostopno 11. 7. 2016].